Thirty Five Dollars

Cromemco

CROMIX

Multi-user
Multi-tasking
Operating System

Instruction Manual

Cromemco®

CROMIX*

Multi-user Multi-tasking Disk Operating System

Instruction Manual

CROMEMCO, Inc.
280 Bernardo Avenue
Mountain View, CA. 94043

Part No. 023-4022

December 1982

This manual was produced using a Cromemco System Three computer with a Cromemco HDD-22 Hard Disk Storage System running under the Cromemco Cromix™ Operating System. The text was edited with the Cromemco Cromix Screen Editor. The edited text was proofread by the Cromemco SpellMaster™ Program and formatted by the Cromemco Word Processing System Formatter II. Camera-ready copy was printed on a Cromemco 3355A printer.

TABLE OF CONTENTS

INTRODUCTION

When microcomputers were first introduced, the most common memory modules contained about 4,000 bytes of storage. Now, 16 times as much memory is available in a module and today's microcomputers also utilize the new technology incorporated in fast hard disk mass storage devices.

The Cromix Operating System was developed by Cromemco to take full advantage of the large amount of random access memory (RAM) and fast hard disk storage available on today's and tomorrow's microcomputers. The Cromix Operating System has many capabilities found only in large mainframe operating systems - capabilities such as:

1. Support of multiple tasks and multiple users on hard disk and floppy disk file storage systems,
2. Multiple hierarchical directories and subdirectories,
3. Device compatible I/O which supports user redirection of input and output,
4. Versatile Shell program for flexible and reconfigurable user interface,
5. Password security system, limiting system and file access as well as protecting files with read, write, append, and execute attributes,
6. Date and time support,
7. Numerous file buffers for high speed execution, and
8. Resident execution of tasks (i.e., jobs are not swapped out to disk) and servicing of users through bank selection for rapid context switching.

A Cromemco customer has a choice of using either the CDOS or Cromix Operating System on Cromemco microcomputers. CDOS has the advantage of years of testing by thousands of users. It is a time proven system. In addition, CDOS has the advantage of being compact in memory utilization. It can reside in the same 64 Kbyte memory board as the user's program. Only 64 Kbytes of RAM memory is required for CDOS and CDOS uses only about one fourth of that memory, with the rest available for programming languages and user programs.

The Cromix Operating System requires 64 Kbytes of RAM for the operating system. Each concurrently executing program normally requires an additional 64 Kbytes, of which only 1K is used by the operating system. Unlike CDOS, the Cromix Operating System supports more than one directory, user, and task. It also offers password security and provides a user interface as well as I/O which may be reconfigured. Additionally, the extensive buffering of the Cromix Operating System makes disk-intensive execution more than twice as fast as CDOS. CDOS offers limited buffering because of its memory limitation.

Some of the Cromix features may not be familiar to many computer users. CDOS may have all of the features that users expected before the advent of the Cromix Operating System. It may be difficult to imagine the ability of the Cromix Operating System to print a file at the same time as the user is editing another file. The Cromix Operating System allows you not only to print, but to execute multiple jobs from one or several terminals at the same time. This multi-processing is commonplace on large mainframe computers, as are the time and security features of the Cromix Operating System.

The ability to allow the user to reconfigure the I/O and user interface is not a common feature on large mainframes. With the Cromix Operating System, disk files may be used in the place of keyboard input for preprogrammed responses to standard programs. Disk files may also be used to store program output which is normally sent to the user terminal screen. If a user does not expect to be present at the terminal during execution of a program, the output may be redirected to a disk file for later viewing. The user interface may also be radically changed when using the Cromix Operating System. Usually an operating system does not allow the user to change commands, but the Cromix Operating System has a programmable Shell which facilitates user interface customization. Thus, a user should expect to use the Cromix Operating System to increase productivity by utilizing the computer's

ability to perform multiple tasks at the same time. Some users will find that either the greater disk throughput of the Cromix Operating System or the support of multiple directories and subdirectories alone justifies its use. For whatever reason the Cromix Operating System is chosen, the user will have access to features that are truly at the state of the art of operating systems and yet are easy to learn and use.

Chapter 1

GETTING STARTED

This chapter is an introduction to the Cromemco Cromix Operating System for the first-time user. By progressing through a sample session, many of the important features of the Cromix Operating System are highlighted. You are encouraged to go through this chapter while sitting in front of a terminal and to expand on the examples given. By doing this several times you should arrive at a level of competence that allows a fuller understanding of the rest of the manual.

Initial hardware and software setups are covered in Chapter 6 and the appendices. It is assumed here that the hardware is set up and functioning properly and that the user has been assigned a user name.

DEFINITIONS

Before discussing the operation of the Cromix system, some important terms must be defined.

Hardware

The hardware is the physical, touchable part of the computer. It is the computer as it exists when no power is applied to it. The parts of the hardware visible from the outside of your computer are the terminal, the printer, and the box containing the computer itself. Inside the computer is more hardware, comprised of disk drives, memory boards, a disk controller board, a central processing unit, and various other boards. It is not necessary for the user to understand the internal functioning of the hardware in order to use the computer.

The hardware is useless without software.

Software

The software is comprised of programs that run on the hardware. The software cannot be seen or touched. It is the software that causes the computer to perform whatever function you have asked it to perform. Commonly used software (or programs) are the Cromemco Screen Editor, Cromemco Formatter II, and Cromemco 32K

Structured Basic. The Cromix Operating System itself is a large program.

Operating System

An operating system is one of the programs running on a computer. The operating system's function is to keep everything within the computer operating smoothly. It is the operating system's responsibility to allocate memory as it is needed, keep track of who is using what space on the disk, and to allow the user to run a selected program or utility. The operating system asks you to log in, asks you for a password, and checks that you have given a valid user name and password. Thus, the operating system allows you to gain access to the computer and communicate with it.

File

A file is a grouping of related information. If you are using the system to do word processing, a file might be a letter that you are composing. If you are writing a program in Basic, you can store your program in a file.

A file in the Cromix Operating System is very similar to a file found in a drawer of a file cabinet. Both contain related information and both are stored under a single name so they can be easily found. Just as you can add to or take from a file in a file drawer, you can change a file in the Cromix Operating System. The difference is that you cannot touch a file stored in a computer. Things may be added to or taken away from the file through use of an editor or other program.

LOGGING IN TO THE SYSTEM

Logging in is the process of informing the computer that a user wishes to use the computer; the operating system responds by acknowledging the authorized user. Because the Cromix Operating System can serve many different users and each user may have access to a unique set of files, a valid user name must be presented to the system before the user can be logged in. Please refer to the description of the **Passwd** utility in Chapter 9 if it is necessary to establish a new user name.

For this example assume the user's name is **fred**, and that fred has the password **mountain**.

When you sit down at the terminal, the first thing that the Cromix Operating System asks you to do is to identify yourself. By displaying a prompt, the operating system tells you it is waiting for you to type in your user name. The prompt is **Login:**. You, in turn, tell the operating system you have finished responding to its prompt by typing your login name followed by a RETURN. To type a RETURN, press the key on your terminal labeled RETURN. Notice that the operating system does not consider anything you have typed until it receives the RETURN character.

The RETURN character is sometimes referred to as a carriage return, a newline, or <CR>.

For this example, the user enters the user name **fred** followed by a RETURN in response to the Cromix Operating System **Login:** prompt. After you have entered your user name, the system requests your password. If you do not have a password, you are not asked to enter one. As is the case with all Cromix Operating System commands, the password must be followed by a RETURN character.

Login: **fred RETURN**
Password:

Notice that the secret password is not displayed as you enter it on the terminal. After the password and the RETURN are entered, the Cromix Operating System responds:

Logged in fred Jun-24-1980 17:12:15 on tty1
%

Throughout this manual, messages and prompts displayed by the Cromix Operating System are in normal type, while responses supplied by the user are typed in **boldface** characters. Pressing the RETURN key on the terminal is represented by RETURN. Thus, in the first example above, the operating system displayed the prompt Login: and the user supplied the response **fred** followed by a RETURN keystroke.

After the first few examples in this manual, you are not reminded that it is necessary to type a RETURN after each command. Remember, if the system does not seem to be responding, you may have neglected to enter RETURN after a command.

After receiving a valid user name and password, the Cromix Operating System displays the message of the day **(Motd)** and a prompt. Normally, the prompt is either a percent sign (%) or a pound sign (#). The Cromix prompt indicates that the operating system is waiting for further instructions.

IMPORTANT NOTES

CNTRL-Q

The Cromix Operating System is set up so that information does not scroll off the terminal screen before the user has a chance to review it. When the screen is full, the terminal emits a **beep.** Enter a **CNTRL-Q** when the information on the screen is no longer needed, so that another screen full of information may be displayed. Enter a **CNTRL-Q** by holding down the **CNTRL** (**CTRL** on some terminals) key and simultaneously typing **q.**

The **CNTRL-Q** feature is disabled by running the **Mode** utility as follows:

% **mode -pa**

Remember, if the terminal seems to have locked up, type **CNTRL-Q.**

CNTRL-C

If you want to stop the computer from doing whatever it is doing, type **CNTRL-C.** Do this by holding down the **CNTRL** key and striking the **c** key. This causes the Cromix prompt to be displayed on the screen.

LOGGING OFF OF THE SYSTEM

A user may log off of the system by entering **ex** or **exit** in response to the Cromix system prompt.

EDITING FILES

As a first exercise, create a file containing a list of names. Use the Screen Editor, which is discussed very briefly here. For further information, refer to the Cromemco Screen Editor manual (part number 023-0081) and the description of the Screen Editor in Chapter 9 of this manual.

The following command causes the operating system to load the Screen Editor and create a file named **friends**:

```
% screen friends
```

If everything is working properly, the banner for the Screen Editor is displayed momentarily and the terminal screen is cleared. The Screen Editor prompt appears across the top of the screen.

Insert a list of names in the **friends** file. To do this, once the Screen Editor is called, type **i** (for insert) followed by the desired list of names, terminating each name with a RETURN. Press the **ESC**ape key to tell the Screen Editor you are finished inserting text. Finally, the command is given to exit from the Screen Editor and write (update) the **friends** file to the disk by typing the characters **e** (for exit) and **u** (for update). The number of characters written to the **friends** file is displayed; followed by the Cromix prompt.

The **Type** command may be used to display the file:

```
% type friends
```

FILE SYSTEM STRUCTURE

The Cromix file system may be thought of as an upside-down tree. At the top of the tree is the **root** and coming down from the root are the **branches**. Some of the branches have additional branches as offshoots and some do not. Note that the tree has no trunk; the branches grow directly out of the root.

Node is the term used to refer to those places on the tree where a branch separates into one or more additional branches. Node is also used to refer to the tips of the branches. In the Cromix file system, every node has a name.

Having established a tree and having named each of the nodes of the tree, it may become necessary to give someone directions to climb out to a specific branch of the tree. The directions start the climber at the location where two or more branches separate from the root. This location is still called the root. From here, we direct the climber to a node. From that node, the climber may be directed to an adjacent node. The climber can climb only between nodes connected by branches. This process continues until the climber reaches the desired node. Using this method, we can instruct the climber to move to any tip of a branch or any intersection where one or more branches are joined.

The instructions can be simplified into a list of nodes given in the order in which the climber reaches them. The term **pathname** refers to this list of nodes.

Two further points make the analogy complete. First, nodes having additional branches coming off them are called **directory** nodes. A directory node has a name, as do all nodes. In addition, a directory node contains a list of the names of all nodes found at the end of its branches, thus the term directory.

Second, the nodes at the ends of the branches are called **ordinary** nodes or ordinary files.

And so the analogy ends. The tree is the file structure the Cromix Operating System uses to store files. The root is the root directory that is always present. Under the Cromix Operating System, the root directory is named /. The directory nodes contain pointers to other directories and to ordinary files. The user stores information in these ordinary files. The ordinary files may contain programs, text, or data.

The Cromix Operating System locates a given directory or ordinary file through use of a pathname. A pathname used to locate a directory is called a **directory pathname.** A pathname used to locate an ordinary file is called a **file pathname.**

Pathnames

Although pathnames do not need to start with the root directory, the current discussion is confined to this type of pathname. A pathname traces a path from the root directory through any intermediate directories to the desired directory or file. For example, the file pathname for the file **motd** is:

/etc/motd

The first / in the file pathname refers to the root directory. Each subsequent / in a pathname separates entries of the pathname. The next entry in the pathname above is **etc**, another directory. Another / separates this directory from an entry in the directory. This entry is **motd**, an ordinary file. (Refer to Figure 1-1.)

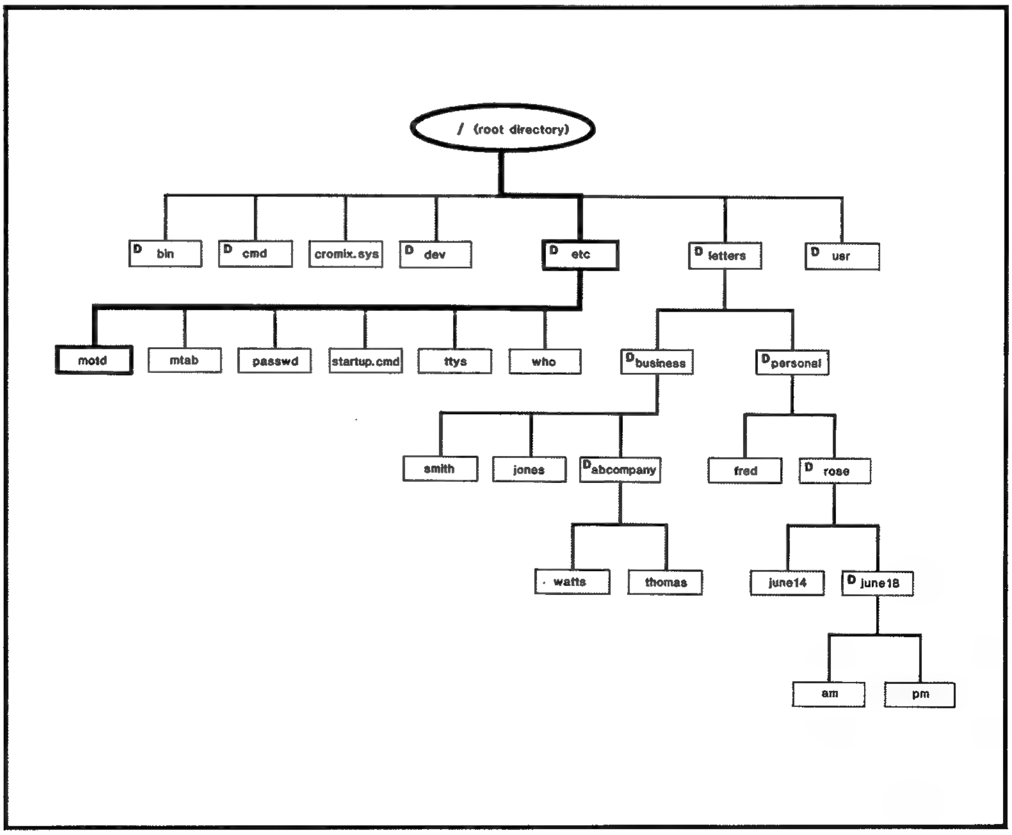


Figure 1-1

If a filename is included in a pathname, it must be the last entry in the pathname. This is called a **file pathname.** Refer to Figure 1-2 and trace the following file pathname:

/letters/business/abcompany/thomas

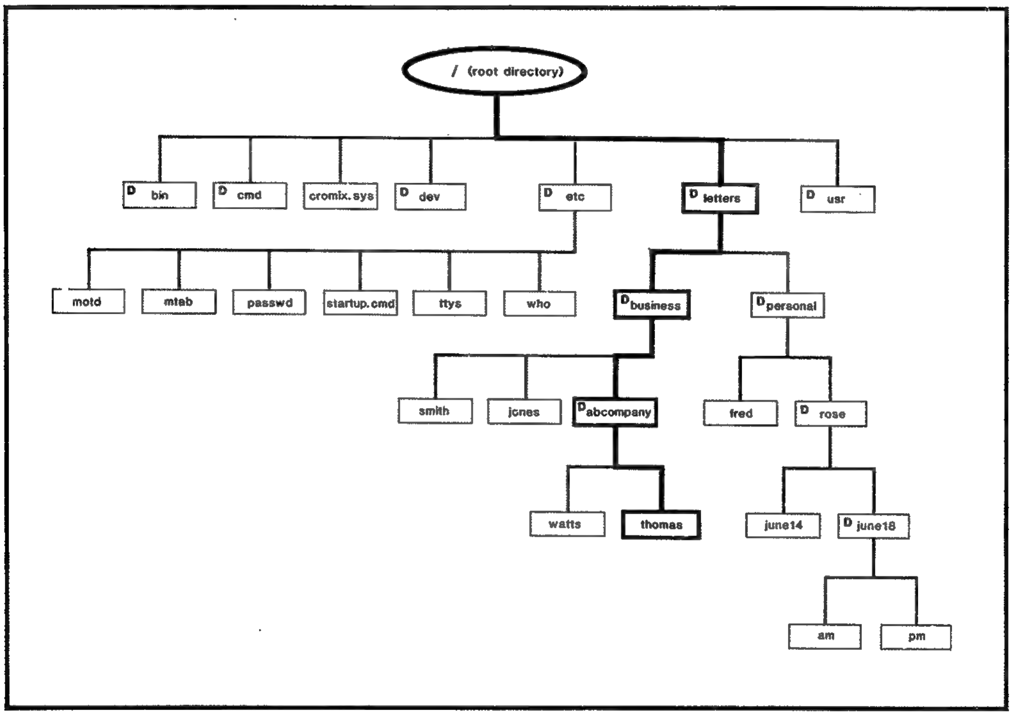


Figure 1-2

Refer to Figure 1-3 and trace the following directory pathname:

/letters/personal/rose/june18/am

A file pathname may be used wherever the Cromix Operating System expects a filename. Similarly, a directory pathname may be used anywhere a directory name is expected.

Current Directory

The current directory specifies those files and directories which may be accessed by giving only a file or directory name (i.e., no pathname is needed). The user has immediate access to the current directory; any other directory must be explicitly specified on the command line.

The current directory can be thought of as another directory from which a pathname may be started. The advanced user is referred to the discussion of **relative pathnames** in Chapter 3.

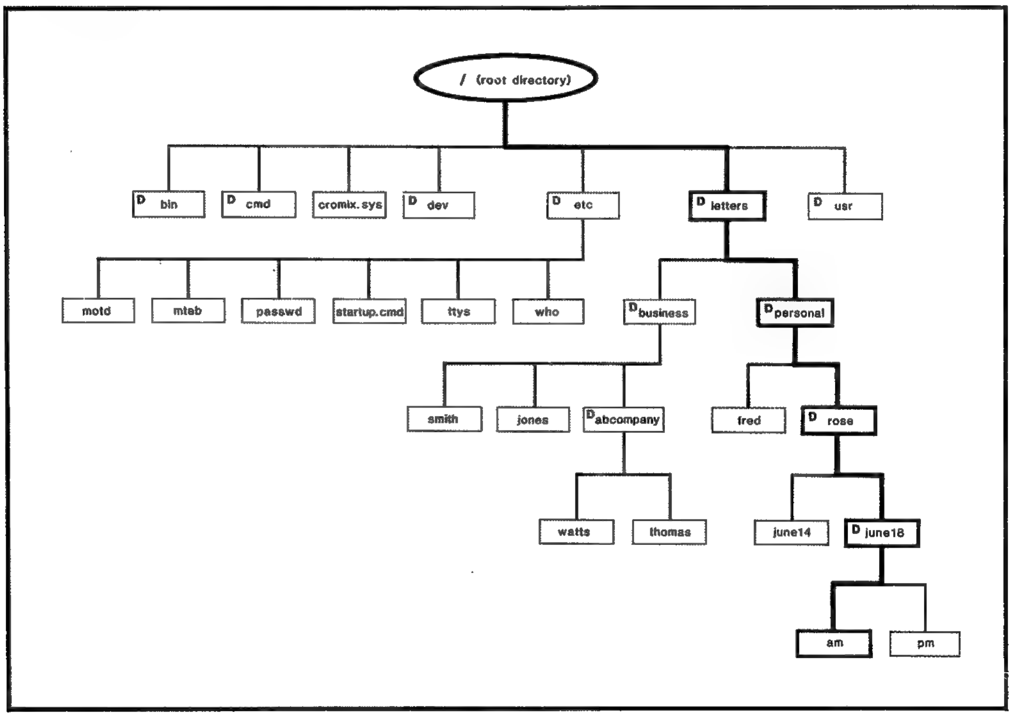


Figure 1-3

CDOS files can take limited advantage of the Cromix file structure. Please refer to the **Sim** utility program for additional information.

Chapter 2

BASIC COMMANDS AND UTILITIES

This chapter covers the basic commands and utilities required to use the Cromix Operating System. More complete descriptions of these and other commands and utilities are given in Chapter 9.

DIRECTORY COMMAND

When the **Directory** command is given, the Cromix Operating System displays the name of the current directory. The **Directory** command is abbreviated as **d.**

```
% d
/
%
```

In the example above, the operating system displayed the name of the current directory by displaying **/**, the name of the root directory. (Do not forget to type RETURN after each command.) The **Directory** command can also be used to change the current directory:

```
% d /etc
% d
/etc
%
```

The Cromix Operating System does not acknowledge the successful completion of the command. In the example above, the user changed the current directory and then entered the **Directory** command to check if the current directory had been changed. The Cromix Operating System responded by displaying the name of the current directory, **/etc.**

L UTILITY

The **L** utility displays an alphabetical list of entries in a directory. Below is an example of the use of the **L** utility. When you use **L**, your display will differ from this because the sizes and names of your files will be different.

```
% l
        52   D  1 bin
        13   D  1 cmd
    36,864      1 cromix.sys
        32   D  1 dev
         7   D  1 etc
        32   D  1 letters
         6   D  1 usr
```

In response to the **L** command, the operating system lists all subdirectories and ordinary files contained in the current directory.

The **L** command displays four columns of information. The column on the left is the number of bytes the file occupies or, if the entry describes a directory, the number of files within the directory. The second column is blank if the entry is an ordinary file and contains a **D** if the entry is a directory. The third column indicates the number of links to the given directory or file. (Links are discussed in another section.) The column on the far right contains the name of the entry, whether a directory or an ordinary file.

There are several ways to make the Cromix Operating System list the entries within a given directory. First, using the **Directory** command, make the directory in question the current directory, and use the **L** utility to list the contents of the current directory:

```
# d /etc
# l
   .
   .
   .
```

Another way to list the entries in a directory (not the current directory) is to type the **L** command followed by a directory pathname:

```
# l /etc
    .
    .
    .
```

The **/etc** directory is listed and when the list is finished, the current directory is the same as before.

MAKE DIRECTORY UTILITY

The first step in establishing the part of the file system shown in Figure 1-3 is to create the necessary directories. To do this, use the **Makdir** (make directory) utility.

```
% makdir /letters
% makdir /letters/personal
% makdir /letters/personal/rose
% makdir /letters/personal/rose/june18
```

In the example above, four new directories are created. Each of these directories is a subdirectory of the previously created directory.

With careful planning, this type of file structure allows you to organize great numbers of files to make each file readily accessible.

Use the Screen Editor to create a file named **am** located in the directory named **june18:**

```
% screen /letters/personal/rose/june18/am
```

If you are doing quite a bit of work in a particular directory, it is easier to change the current directory rather than specify a long pathname every time a file is used. Enter the **Directory** command with the desired directory pathname:

```
% d /letters/personal/rose/june18
% d
/letters/personal/rose/june18
%
```

In the example above, the **Directory** command is used first to change the current directory and then to display the pathname of the current directory. It is easier to call the Screen Editor and create the file **am** in the current directory:

% **screen am**

If you are not familiar with the Screen Editor, refer to the Cromemco Screen Editor Manual.

All directories specified in a pathname must have previously been created with the **Makdir** command. The Cromix Operating System does not automatically create directories.

TYPE COMMAND

Assume the file **am** exists as specified in Figure 1-3. The **Type** command may be used to display the contents of the file. The **Type** command is abbreviated **ty**.

% **ty /letters/personal/rose/june18/am**

Using the full file pathname (starting with /, the root directory), a file in another directory may be reviewed or edited without changing the current directory.

RENAME COMMAND

The **Rename** utility changes the name of a file. Follow the **Rename** command, abbreviated **ren**, with the existing name (or pathname), a space, and the new name (or pathname). For example:

% **ren fred joe**

or

% **ren /letters/business/jones /letters/business/william**

In these examples, the file **fred** is renamed **joe**. Because the file is in the current directory, no

pathname is used. The second example renames a file which is not in the current directory. The name of the file is changed from **jones** to **william.** Because the file is not in the current directory, the entire file pathname is required.

DELETE COMMAND

The **Delete** utility removes a file. Follow the **Delete** command, abbreviated **del,** with the name (or pathname) of the file or directory to be deleted. To delete a directory, all files in the directory must have been previously deleted and it must not be the current directory. Once the contents of a file are deleted, they cannot be recovered.

```
% del joe
% del /letters/business/william
```

In these examples, the files renamed using the **Ren** command were deleted.

Chapter 3

ADVANCED FEATURES

This chapter describes some of the advanced features of the Cromix Operating System.

TREE DATA STRUCTURE

A **tree** is a data structure. A data structure is a method of storing data or information for easy access. The tree data structure is the inverse of a natural tree. As shown in Figure 3-1, the Cromix file system is an example of a tree data structure.

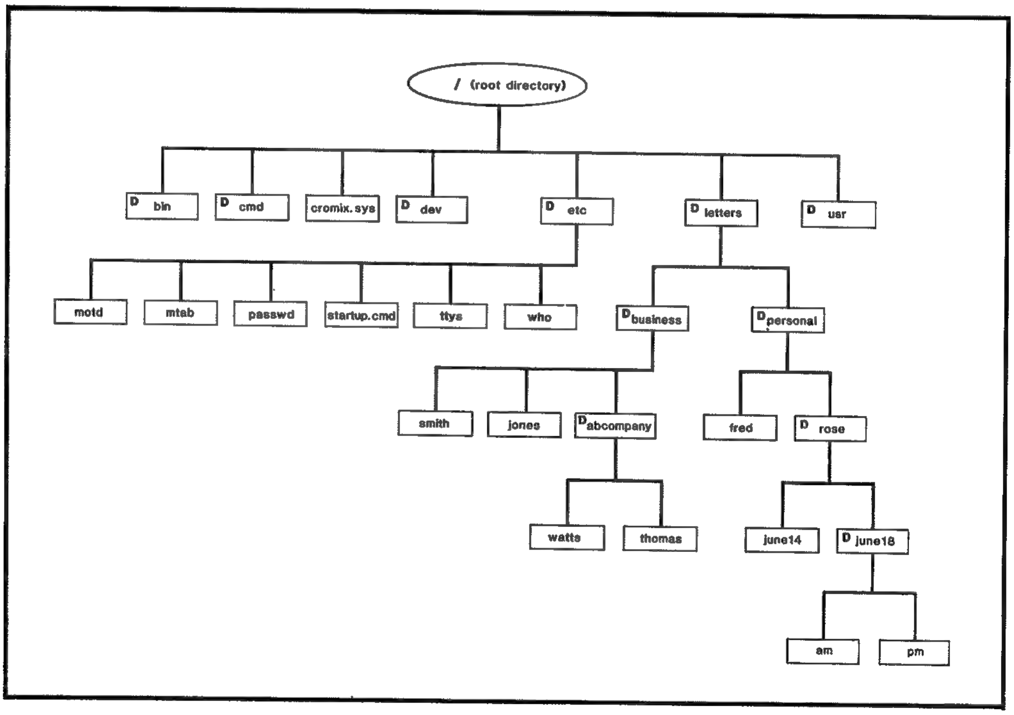


Figure 3-1

The **root** is at the top and the **branches** grow down from the root. In addition to branches and a root, the tree has **nodes.** The term node applies to any of the places on the tree where one branch divides into two or more branches, and to the root and the ends of all of the branches.

Another key idea in the concept of a tree is that of **ancestors** and **descendents.** All nodes are descendents of the root node, or, the root node is the ancestor of all nodes. A direct descendent or ancestor is a node directly connected (by a branch) to another node. Referring to Figure 3-1, **motd** is a direct descendent of **etc.** The same relationship may be looked at by calling **etc** a direct ancestor of **motd.** The terms parent and child may be substituted for ancestor and descendent.

CROMIX FILE STRUCTURE

The Cromix file structure is a tree whose nodes are composed of directories of ordinary files and other (descendent) directories. The highest level directory is called the **root** directory.

The following table defines some of the terms used to discuss directories, files, and devices.

TYPES OF DIRECTORY ENTRIES

Directory — A directory that appears as an entry in another directory is a **descendant directory.**

Data File — A data file that appears as an entry in a directory is an **ordinary file.**

Device

Character Device
A character device is a sequential access device (e.g., terminal, printer, etc.)

Block Device
A block device is a random access device that can maintain a file system (i.e., a disk).

. — A single period refers to the directory in which the entry occurs. The period is not a directory name but a reference to a directory. This reference always assumes the value of the **current directory.**

.. — Two periods refer to the **home directory.** This is your current directory when you first log in.

^ A caret (up-arrow) refers to the ancestor directory. There is no ancestor for the root directory. The caret is not a directory name but a reference to a directory. This reference assumes the value of the **ancestor directory.**

PATHNAME

A **pathname** locates a file or directory within the file structure. The simplest form of pathname is a filename. If a filename is specified and no directory name is given, the file is assumed to be in the current directory.

All names within a pathname are separated by slashes (/). Each succeeding directory name, and the final directory or filename, must be a descendent of the previous directory.

Only directories appear in a directory pathname. If the pathname is a file pathname, the last item is a filename.

A pathname may contain a maximum of 128 characters. The first entry in a pathname must be one of the following:

/
.
^
directory name
filename

The caret (indicating an ancestor directory) may only appear as the first entry in a pathname and may be followed by one or more additional carets. Each successive caret indicates another generation ancestor directory. Multiple carets are not separated by slashes. If an attempt is made to specify a directory as an ancestor of the root directory, the Cromix Operating System proceeds as though the root directory had been specified. See examples in the **Relative Pathname** section below.

Absolute Pathname

An **absolute pathname** locates a file or directory relative to the root directory. This type of pathname always begins with a slash (/) to indicate the root directory and may be followed by any number of directory names.

The following examples refer to Figure 3-1 and make no assumptions about the current directory.

/cromix.sys refers to the file **cromix.sys** located in the root directory

/etc/who refers to the file **who** located in the directory **etc**, which is located in the root directory.

/letters/personal/fred
refers to the file **fred** located in the directory **personal**, which is located in the directory **letters**. The directory **letters** is located in the root directory.

An absolute pathname must be unique. If this weren't the case, the system would not know which of two files with the same absolute pathname you were referencing.

Filenames and directory names may be duplicated, as long as the duplicate names do not appear in the same directory. In this case it is always possible to distinguish between two files by using an absolute pathname.

Relative Pathname

A **relative pathname** locates a file or directory relative to the current directory. The caret (^), indicating the ancestor directory, is useful in defining a relative pathname. Note that the slashes in a relative directory are used as delimiters and do not refer to the root directory.

The following examples refer to Figure 3-1, and assume the directory **personal** is the current directory:

^/business refers to the directory **business** located in the ancestor directory **letters**.

^/business/jones
refers to the file **jones** located in the directory **business,** which is located in the (ancestor) directory **letters.**

rose/june14 refers to the file **june14** located in the descendant directory **rose.**

Assume the root directory is the current directory:

etc/motd refers to the file **motd** located in the directory **etc.** The directory **etc** is located in the root directory.

Current Directory

Up to this point, we have said that files in the current directory can be referenced without the use of a pathname. Strictly speaking, this is not true. A file in the current directory can be referenced without the use of an absolute pathname.

A filename itself is actually a relative pathname that references a file located one level below the current directory. Assuming that **personal** in Figure 3-1 is the current directory, the file named **fred** may be referenced simply as **fred.** In this case, the file reference **fred** is a relative pathname for a file located one level below the current directory. Thus, the file **fred** is a direct descendent of the current directory. For the sake of simplicity, this manual refers to a file that is a descendent of the current directory as being in the current directory.

FILE PROTECTION

The Cromemco Cromix Operating System offers protection for files on many levels.

All files may be opened for exclusive or nonexclusive access. A file opened for exclusive access may not be opened by another process until it is closed by the process that originally opened it. If a file is opened for nonexclusive access, it may be simultaneously opened and accessed by more than one process.

File access privileges are divided into three population segments and four types of file accesses.

The first population segment is the **owner** of the file. This is normally the creator of the file. The second population segment is the **group** to which the owner belongs. A user's group number can be verified in the **/etc/passwd** file. The third population segment is the general **public.** This segment includes all system users.

There are four types of file access for each population segment. The first is **read** access. Read access allows the designated user to read the file. If a user has read access for a directory, the user may list the contents of the directory.

The second is **execute** access. Execute access allows the user to execute the file. If the user has execute access for a directory, the user may use the directory in a pathname.

The two remaining types of access are **write** access and **append** access. Write access allows the user to write to the file, meaning the user may write over or change data in the file.

Append access allows data to be added to the end of the file. Data may then be written to the file at a point **past** the end of file and the end of file indicator is moved to the end of the newly added data. If append access is the only access specified, data written to the file may not be read.

Append access does not imply write access, but write access implies append access.

One type of access privilege for a population segment does not imply any other access privilege for that population segment. The categories of access privileges are combined to provide meaningful data handling. For example, a user with write access to a file normally has read access.

One important point to consider when determining file access privileges is that the file's owner is a member of a group and a member of the public. Implicitly, the user has all access privileges granted to the public and to the group. Any member of the group enjoys all access privileges granted to the public.

All files are created with default access privileges as follows: read, execute, write, and append access privileges for the owner; read and execute access privileges for the group and public. The default owner is the user name of the user who executed the command creating the file. The system gathers its information

on user name from the **/etc/passwd** file. (This default may be changed by generating a new operating system with the **Crogen** utility. Refer to Chapter 9 for more information.)

When files are created by programs that a user is running (e.g., Screen, Debug, WriteMaster, etc.) those files take on default attributes as described above. These same programs can also alter existing files. In this case, the owner name is unchanged but file access attributes may change. For instance, the Screen Editor does not change file attributes after an editing session but, using the **w** command within the Debug program changes all file attributes to the default values. Since this effect may vary from command to command within a single program, and from program to program, users should be aware that file attributes are not immutable.

Access privileges take on a different meaning when applied to a directory. Read access for a directory means the user can use the **L** utility to see the contents of the directory. Execute access means the directory may be used in a pathname or that the user has access **through** the directory. Write access means the user can alter the directory.

The **L** utility program with the **-l** option may be used to check the access privileges associated with a given file. For example, the following command will list the access privileges of file **xyz**:

```
% l -l xyz
312 1 rewa re-- re-- joe Mar-09 18:25 xyz
```

Reading this display from left to right, two items precede the access information: the numbers of bytes in the file (312) and the number of links to the file (1). The access information is displayed as three clusters of four characters. The four characters are **r** (read), **e** (execute), **w** (write) and **a** (append). The presence of one of these characters indicates that the specified population segment is endowed with the specified access privilege. The population segments are, from left to right, **owner**, **group**, and **public**. Thus, in the above display, the owner has all four access privileges while the group and public have only read and execute access privileges.

The last four items in the preceding display are the name of the owner of the file (joe), the date and time the file was created, and finally, the name of the file.

Users working within the Cromix file system must explicitly check the access attributes of files and directories they work in, use, or create for other users. Users must be aware of accessory files that may be required by programs they are running -- help files, libraries, and so forth. Access and ownership of the accessory file -- and access and ownership of parent directories all the way to the root -- must be compatible with the operation of the program being executed.

For all errors implying access limitations always check access privileges and the ownership of the directories and files involved, and of all ancestor directories.

Chapter 4

NAMES

This chapter covers names used within the Cromix Operating System. File, directory, and ambiguous filenames are discussed, as well as file naming conventions.

FILE AND DIRECTORY NAMES

Any name within the Cromix Operating System (including file, directory, and device names) may contain from 1 to 24 characters from the following set:

A-Z a-z 0-9 $ _ .

The Cromix Shell, which processes commands, does not distinguish between upper and lower case characters in file and directory names. On entry, all names are converted to and stored as null terminated strings of lower case characters.

Invisible Names

If a period is the first character of a name, it is an **invisible name** and is not normally listed with the rest of the directory. Refer to the **L** utility, **-a** option, for more information.

AMBIGUOUS FILE AND DIRECTORY NAMES

The Cromix Shell expands ambiguous file and directory names into a list of existing names that match a specified pattern. These names may be used by any program designed to accept a list of names.

The **asterisk** (*) can substitute for any string of zero or more characters delimited by periods (.) on one or both ends of a string. For example:

```
a*b matches
               ab
               axb
               axyb
               a$$$b
```

and

```
*.z80
```

matches all files in the current directory whose filenames end in **.z80**.

A **double asterisk** (**) substitutes for any string of zero or more characters except strings where the first character is a period. Imbedded and trailing periods are not considered delimiters by the double asterisk. In other words, it matches all names not beginning with a period. For example:

```
** matches all names
```

The **L** utility is used to demonstrate how the Shell expands filenames containing asterisks. If these files:

```
a
a.a
a.a.a
```

exist in the current directory, then the Shell expands the ambiguous filenames *****, ***.a**, **a.***, ***.***, and ****** as shown on the following page:

```
% l *
          49    l a

% l *.a
          62    l a.a

% l a.*
          62    l a.a

% l *.*
          62    l a.a

% l **
          49    l a
          62    l a.a
         111    l a.a.a
```

The **question mark** (?) substitutes for any single character. For example:

```
a?b matches          axb
                     alb
                     a$b

but does not match   ab
                     axyb
                     a$$$b
```

Square brackets([]) indicate that any of the characters contained within the brackets are to be substituted in the string. For example:

```
[abcd] matches
           a
           b
           c
           d
```

A range of characters may be specified by the delimiters of the range separated by a **hyphen** (-):

```
a[xyza-d] matches
               ax
               ay
               az
               aa
               ab
               ac
               ad
```

Quotation marks or apostrophes (" or ') have a special significance to the Shell. All characters appearing between matched sets of either of these two characters are taken literally. This means that the question mark, left and right bracket, the single and double asterisk, and the dash (within the brackets) lose their special significance and that the Shell does not expand them. All characters that appear between a pair of quotation marks or apostrophes are taken as a single argument.

Special characters may be passed to calling routines through the use of quotation marks and apostrophes. This is why many Cromix utility programs require that ambiguous filenames be placed between quotation marks or apostrophes; the utility may then expand the filenames as required.

For example:

% **echo ****

might yield:

```
a
a.a
a.a.a
```

whereas:

% **echo "**"**

would result in the following display:

```
**
```

File Naming Conventions

The Cromix Shell looks for three types of filename extensions and interprets these extensions as having special significance. Some filenames have an extension - a portion of a filename that follows the final period embedded within a filename.

The filename extension **.bin** means the file is an executable file that runs directly under the Cromix Operating System.

The filename extension **.com** means the file is an executable file that makes use of CDOS system calls. The Cromix Operating System automatically loads the CDOS Simulator (**sim.bin**) with this type of file.

The filename extension **.cmd** means that the file is a Cromix Shell program. The Shell interprets each line of a **.cmd** file as a Shell command line.

Special Names

Two special filenames may appear in a user's home directory (the directory a user logs in to). One is **.reminder**; the contents of this file are displayed each time the user logs in. The other is **.startup.cmd**; this command file is executed each time the user logs in. Because both these filenames begin with a period, they are invisible entries.

Chapter 5

THE /ETC DIRECTORY

This chapter discusses the files found in the **/etc** directory. When listed, the directory appears as follows. (Your **/etc** directory will differ.)

```
% l /etc
Directory: /etc
        4096   1 fdboot
          55   1 group
         239   1 iostartup.iop.cmd
        3476   1 login.bin
          69   1 motd
         128   1 mtab
         174   1 passwd
        4096   1 sfdboot
          36   1 startup.cmd
         304   1 startup.msg
         287   1 ttys
         287   1 ttys.iop
         567   1 warning
          48   1 who
```

ACCOUNT

The **account** file (not included above) may be included in the **/etc** directory. When this file is present, information concerning users logging on and off the system is written to it.

A privileged user may use this information to determine system utilization, produce departmental billing and reports, etc.

Records in the account file are 48 bytes long. The first 16 bytes in each record indicate the terminal device on which the user logged in. The next 16 bytes contain the user name. Following this, three bytes contain the date, three bytes the time, and two the user id. The next two bytes contain the group id, one byte contains the tty major device number, and one byte contains the tty minor device number. The last four bytes are reserved for future use. A plus sign (+) in the **login** user name field indicates when the system was booted, or brought up.

The **Who** utility may be used to display the **account** file:

```
# who /etc/account
```

This file should be deleted and recreated periodically, as it grows with system use.

GROUP

The file **/etc/group** must be present for the **Mail** and **Passwd** programs to operate properly with group parameters. This file has a format similar to the **/etc/passwd** file. The following fields appear on each line. The fields are separated by colons. The line is terminated by a RETURN character.

1. Group name

2. Group password

3. Group identification

4. User names of all users associated with the group, separated by commas

The file contains one line for each group. Refer to the **Passwd** utility for information on adding, deleting, or changing groups.

IOSTARTUP.IOP.CMD

This command is used to log in terminals on Input/Output Processor (IOP) and Quadart boards. Refer to Chapter 6 for more information.

MOTD

The **motd** file is the **m**essage **o**f **t**he **d**ay file. The contents of this file are displayed each time a user logs on to the system. A privileged user may edit this file to cause display of any desired message. This is an informational file and contains no commands to the system.

MTAB

The **mtab** file contains the **mount table**. When the **Mount** command is given without arguments, the **mtab** file is consulted and a list of mounted (online) devices is displayed. This file must not be edited by the user.

The mtab file contains one 128-byte record for each disk mounted. The first 32 bytes of each record contain the device name, which is left justified and null padded. The last 96 bytes of each record contain the dummy pathname where the device is mounted. The first record in mtab always specifies the root device.

PASSWD

The **passwd** file contains information about each user. This information includes an encryption of any required password as well as restrictions on the user.

Each line of the passwd file represents one user. Each line has six fields separated by colons.

The first field is the user name. This is the name that must be typed in response to the Cromix Operating System prompt **Login:**. The second field is an optional encrypted password. Refer to the **Passwd** utility for information on adding, deleting, or changing passwords.

The third and fourth fields are the user and group identification numbers. Each of these fields is an unsigned integer between 0 and 65535. A zero in the user field indicates a privileged user. A zero in the group field indicates that the user is not a member of any group. Any other number only has significance within a given system.

The fifth field is the initial, or home, directory. This is the user's current directory immediately upon logging on. The last field is an optional command line. If this line is blank, the user may run the Shell program. If any other command line appears here, execution of the command line begins automatically when the user logs on, and the user is logged off automatically when execution of the command line terminates.

Cromix Operating System version 11 has a very secure password encryption scheme. Since the password encryption differs from previous releases of the Cromix Operating System, a user WILL NOT be able to log in on a

disk under the new Operating System (version 11.00 or higher) unless the **/etc/passwd** file contains at least one privileged user name without a password. Refer to Chapter 6 for more information.

STARTUP.CMD

The **startup.cmd** file contains Shell commands that are executed when the system is started up. As shipped, this file contains a command to execute the **Time** program used to set the system clock and date.

TTYS

The **ttys** file contains a list of all possible terminals and pertinent information for each terminal. This file must be edited using the Screen Editor to change the number of terminals that may be attached to the system.

Each line in this file represents one terminal. The first entry on each line is a **1** or **0**. A **1** indicates that the terminal is present, a **0** indicates that it is not.

The next column is delimited by a colon and represents the baud rate of the terminal. The baud rate for any one of the terminals may be one of the following: 19200 (except the system console), 9600, 4800, 2400, 1200, 300, 150, 110, N, or A. **A** indicates that the baud rate is automatically established when the user presses RETURN several times. **N** indicates no change in the baud rate. The system console must **not** be set to a baud rate of 19200 if it is connected to the 4FDC or 16FDC.

The third column is delimited by a colon and contains the name of the terminal. The terminals are named tty1 through tty9, qtty1 through qtty64, and mtty1 through mtty64.

WHO

The **who** file contains information on all users currently logged on the system. The format of the **who** file is identical to that of the **account** file.

The **Who** utility may be used to display the contents of the **who** file. Please refer to Chapter 9 for additional information.

Chapter 6

BOOTING AND SETTING UP THE CROMIX OPERATING SYSTEM

This chapter explains the procedure for booting, or bringing up a Cromix system. It explains how to create a set of backup system disks, how to set up a Cromix system on a hard disk, and how to convert your system from a single-user to a multi-user system using either TU-ART digital interface boards or an IOP (Input/Output Processor) and Quadart boards.

BOOTING THE CROMIX SYSTEM FOR THE FIRST TIME

The procedure for bringing up the system for the first time from a floppy disk is summarized here:

1. Boot up from the system disk.
2. Login.
3. Make a backup copy of the system disk or disks.
4. Reboot using the backup copy of the disk.

Booting Up From the Cromix System Disk

Cromemco recommends that machines which are used daily be left on continuously in order to reduce stress caused by frequently turning the system on and off. This keeps the components at a constant temperature and reduces the occurrence of transient voltages. First, power up your system, then, insert the Cromix Operating System disk in drive A. The disk must not be write protected. (Diskettes should be removed from the drives when turning the system on and off.) Reset the computer. If you are not using a Cromemco 3102 terminal, you will need to depress the RETURN key on the terminal several times until this message appears:

```
Preparing to BOOT, ESC to Abort
```

If you are using a Cromemco 3102 terminal, the message will appear automatically. The light next to drive **A** will go on and you should hear clicking sounds coming from the drive as the disk is read. If the Resident Disk Operating System (RDOS) prompt (;) appears, type **b**

and press RETURN to boot the Cromix Operating System. Disk drive A should immediately become active as the operating system program is loaded. Once the system starts to boot, the following message appears:

```
Standby
```

After a few seconds the disk light will go out, the terminal will display the new user message, and the operating system will prompt you for the date:

```
Date mm/dd/yy
```

The month, day and year must be separated by slashes (/), periods, colons (:), or spaces. When the prompt

```
Time  hh:mm:ss
```

is displayed, enter the time in the same way. The seconds are optional.

Login

The system now displays the login prompt.

```
Login:
```

As the system is shipped, you may login as **system**, **user1**, or **user2**. Initially, there are no passwords for these login names. Enter **system** to log in as a privileged user. This allows you to edit the **/etc/passwd** and **/etc/ttys** files necessary to set up a multi-user system as well as to execute the **Newdisk** command used to create backup system disks. On a single user system, login under **system** to establish a login name other than **user1** and a password.

After entering the login name, you are given a Cromix prompt, showing you now have access to the operating system.

\# is the prompt for the privileged user **system.**
% is the prompt for **user1**, **user2**, and other nonprivileged users.

Now that you have logged on to the Cromix Operating System, one of your first tasks is to create a backup disk.

Copying the System Disk

Making a backup copy of your Cromix system disk is an essential precaution. Rather than use your original system disk, you can use the copy and keep the original as a backup. If you have only one disk drive, this step must be omitted.

The disk onto which the backup is to be written is inserted in drive B. **This disk must either be blank or expendable since all data on this disk will be destroyed.**

Type the command **newdisk** followed by the device name of the drive housing the disk onto which you are copying your system disk. For example, typing

newdisk sfdb RETURN

creates a copy of the system disk on a 5-1/4 inch floppy diskette in drive B.

Newdisk first executes the Init program, which initializes a disk. Init prompts you for the device designation of the disk to be initialized (See Table 6-1). This is the disk onto which the system is being copied. In our example, the response would be **sfdb.** Remember that all data on the disk specified here is deleted in the initialization process. All other questions from Init can be assigned the default values by pressing the RETURN key in response to the prompts. The default values appear in square brackets [] on the command line.

Table 6-1: DISK DEVICES

Disk Designation	Physical Device	Minor: Major Device Number
fda	large floppy disk A	1:0
fdb	large floppy disk B	1:1
fdc	large floppy disk C	1:2
fdd	large floppy disk D	1:3
sfda	small floppy disk A	1:4
sfdb	small floppy disk B	1:5
sfdc	small floppy disk C	1:6
sfdd	small floppy disk D	1:7
hd0	hard disk E	2:0
hd1	hard disk F	2:1
hd2	hard disk G	2:2

After the initialization questions have been answered, the **newdisk** command automatically executes a series of programs that copy the system disk, and no further user input is required. After several minutes, this process is complete and the message

```
finished creating disk devname
```

is displayed, followed by the Cromix prompt.

Cromemco strongly recommends that the Cromix system disk be copied for use and that the original disk be stored in a safe place.

If you do not wish to read the instructions on copying disks each time the system is booted up, the second line of the file **/etc/startup.cmd** may be deleted using the Screen Editor.

The Cromix system, as supplied on 5-1/4 inch diskettes, includes two disks: the Cromix system diskette (Disk #1) and Disk #2. Disk #2 contains all of the files in the **/usr** directory including the online Cromix manual. Disk #2 is used by mounting it in the Cromix directory structure. Refer to the description of the **Mounthelp** command for details.

Disk #2 may be backed up by inserting it in drive **A** and running **Newdisk**, just as you would for Disk #1. If you are using a large floppy disk, the Help files are included on the system disk and have therefore already been backed up during the **Newdisk** procedure.

Booting from the New System Disk

Now log off the system by entering **ex** in response to the system prompt **#**. Remove the system disk from drive A and store it in a safe place. Move the newly created copy of the system disk from drive B to drive A. Follow the boot procedure which was described above in **Booting Up from the Cromix System Disk.**

UPDATING YOUR CROMIX SYSTEM

The password encryption scheme used in Cromix version 11 is greatly improved and is more secure than that used in previous versions of the Cromix system. Consequently, the **/etc/passwd** file of version 10 Cromix disks **must have all passwords removed** while the previous version of the Cromix Operating System is running. These may be replaced while running the new version of the operating system.

There are many differences between Cromix versions 10 and 11. For instance, all **/bin** files and all device drivers need to be replaced to upgrade the system to version 11. To accomplish the task smoothly and quickly, a new utility, **Update**, is provided with Cromix version 11. This utility can also be used to update any Cromix version 11 to a newer version (such as changing 11.05 to 11.09). To update to version 11, put the new version 11 system disk in disk drive A. Boot from the disk. Type the command:

update devicename

where the devicename indicates the floppy disk or hard disk (see Table 6-1) on which the old system resides. If you are using 5-1/4" diskettes, repeat this procedure with Disk #2 in drive **A.**

The user login procedure has been separated from the main portion of the Cromix Operating System to make more room in the system memory bank. It is now a separate program (**login.bin**) stored in the **/etc** directory. This program must be in the **/etc** directory on all disks from which you will boot Cromix version 11.

SETTING UP A CROMIX SYSTEM ON A HARD DISK

If you have a hard disk on your system, you will want to take advantage of the speed and reliability which it provides. The procedure for bringing up a single user Cromix system on a hard disk is very similar to bringing up a floppy disk system. It is summarized here:

1. Boot up the Cromix system disk.

2. Login.

3. Make a backup copy of the system disk onto a floppy if you haven't previously done so.

4. Copy the System disk onto the hard disk.

5. Using **/gen/default**, set the default root device to the hard disk (refer to Chapter 9).

6. Boot up the system using the hard disk as the root device.

7. Copy Disk #2 onto the hard disk. (This step only applies when using 5-1/4" floppy diskettes.)

Follow the instructions for floppy disks found in **Boot Up from the System Disk**, **Login**, and **Copying the System Disk** for Steps 1, 2, and 3 above. By the time you are ready to continue with Step 4, you are booted up, logged in as **system**, and can see the Cromix prompt on your screen.

Copying the System Disk onto the Hard Disk

The system disk can be copied to the hard disk in much the same manner as it was backed up onto another floppy diskette. Execute the **Newdisk** command by typing

newdisk hd0

where **hd0** is the device name for the first hard disk on the system.

As described earlier under **Copying the System Disk, Newdisk** executes the Init program and prompts the user for the name of the disk to be initialized. Since this is the disk onto which the system is copied, the proper response is **hd0,** the drive designator for the first hard disk (see Table 6-1). Remember that all data on the specified disk is deleted in the initialization process, so any programs or data to be preserved should be copied off the hard disk before proceeding. All other questions can be assigned the default value by pressing RETURN in response to the prompts. **Newdisk** then executes a series of programs and displays the message

```
finished creating disk devname
```

when the process is complete.

If you are using 5-1/4" diskettes, insert Disk #2 in drive **A**, boot the system, and give the command **update hd0**.

The Cromix Operating System now resides on the hard disk and can be booted using the hard disk as the root device instead of the floppy.

A floppy disk is still required as a boot disk, since RDOS does not allow you to boot up directly from a hard disk. The system disk can be used as the boot disk, or you can create one by initializing, making a file system, writing a boot track, and copying the file **cromix.sys** to a new disk.

For example, a large floppy disk inserted in drive B might be prepared as a boot disk using the following commands:

```
init      (respond to all questions)
makfs fdb
wboot fdb
create /b
mount fdb /b
copy /cromix.sys   /b/cromix.sys
unmount fdb
```

In the same way, you can back up your original Cromix master diskette (if you have only one drive) by using the hard disk as the root and typing **Newdisk sfda** or **Newdisk fda.** Once you boot up all files are read from the default root device.

To reboot the system, first enter **ex** to log off.

Booting Up Using the Hard Disk as the Root Device

Execute the command

```
# /gen/default /cromix.sys 2 0
```

to make the system default to hd0 as the root device. If the boot disk is not the root, mount the boot disk and use Default on the **cromix.sys** file on that disk.

When using a hard disk, Default should be used on **cromix.sys** in both the hard disk root directory and the boot disk.

The terminal now displays the new user message and prompts you for the date and time. At this point, the hard disk is used as the root device and drive A is no longer used for any purpose. Since this disk is not the root device, and is not mounted in the Cromix directory structure, it can be removed from the drive and put away.

Note that under ordinary operation, a floppy disk specified as the root must never be removed from its drive, since the operating system must constantly refer to utility and command files on that disk.

Copying the Help Files to the Hard Disk

If your system disk uses a large (8") floppy diskette, all files are included on the system disk. They were copied to the hard disk when the **Newdisk** command was executed. If your system uses 5-1/4" floppies, the **/usr** directory, including the **help** files are included on a second diskette, Disk #2. In this case, these files must be copied to the hard disk, using the **Update** utility. Insert Disk #2 in drive **A,** boot the system, and give the command **Update hd0.**

USING A PRINTER - SOFTWARE CONSIDERATIONS

Output sent to the system printer is displayed on the device to which the **/dev/prt** file is linked.

Dot Matrix Printer

The Cromix Operating System is shipped assuming a dot matrix printer (Cromemco Model 3715 or 3703) is attached to the system. If the software has been altered, a dot matrix printer may be specified as follows:

```
# maklink -f /dev/lpt1 /dev/prt
```

Fully Formed Character Printer

The following command specifies that a fully formed character printer (Cromemco Model 3355B) is the system printer:

```
# maklink -f /dev/typ1 /dev/prt
```

Serial Printer

A serial printer driver is available under the Cromix Operating System. This driver utilizes an XON/XOFF protocol. The serial printer(s) may be connected to any physical port where a terminal can be connected; that is, 16FDC, TU-ART, or Quadart. Appendix D shows the relationship between device numbers and port addresses. Of course, you cannot connect a terminal and a serial printer simultaneously to the same physical port.

Three steps are required to connect a serial printer to a Cromix system.

1. Execute **Crogen,** including drivers for either TU-ART or Quadart printer drivers as necessary, then give the **Boot** command, specifying the desired **cromix.sys** file.

2. Connect the serial printer(s) to TU-ART or Quadart ports not used by a terminal and whose ports (minor device number) do not conflict with an existing serial device.

3. Make a device name for the printer using **Makdev.** The table of device names for serial printers is in

Appendix D of this manual. Use the device name whose minor device number corresponds to the base port address of the port to which the printer is connected. For example, the printer may be connected to the extra TU-ART port on a two user Cromix system - TU-ART B, base address 50h. The command line to make the device name would be:

```
#makdev slpt3 c 7 5
#chowner bin slpt3
```

The last line is necessary, because all device files in the Cromix version 11 must be owned by **bin.** If they are not owned by **bin,** it may be possible for the user to print, but **Spool** may not work properly.

Notes

The **/dev** directory may only be altered by a privileged user (such as **system**).

The file **$LP** in Cromemco 32K Structured Basic and 16K Extended Basic is associated with the system printer. As such, all output to the **$LP** file goes to the device linked to the **/dev/prt** file. This is also true of the standard **LUNs** in FORTRAN and **PRINTER** in COBOL.

An option has been added to the SLPT and QSLPT drivers.

As was previously the case, the driver uses the XON/XOFF protocol if the minor device number is less than 128. For instance, if the printer sends the driver an XOFF character, the driver suspends printer output until an XON character is received.

However, if the minor device number is at least 128, the driver uses an ETX/ACK protocol. After sending 60 characters to the printer, the driver normally sends the printer an ETX character. The driver suspends further output until it receives an ACK character from the printer. The exception to this rule is described below.

Some printers reserve certain character sequences for use as command sequences. For example, sending an ESC character followed by ; to certain printers sets the width to 132 columns. The driver must not send an ETX character in the middle of one of these command sequences.

The driver contains a pair of tables which describe the

command sequences of the printer. These tables may be located by searching for the ASCII string **SEQ** immediately preceding the hexadecimal address of the second table. The first table follows this address.

The first table describes the characteristics of the first character of a command sequence. The second table describes the characteristics of the second character. Each table entry is comprised of an ASCII character followed by a data byte. The byte 80h denotes the end of the tables.

Command sequences may be 2- or 3-characters long, or of indefinite length. The last are terminated by an ASCII NULL (00).

An entry's data byte determines the characteristics of the command sequence according to the following bits:

```
CMDSEQ      equ  4  ;this character is part of a command
                    ;sequence
CSBEGIN     equ  5  ;beginning of a command sequence
CSNXLAST    equ  6  ;this is the next-to-last character
CSLAST      equ  7  ;this is the last character
```

The command sequenced tables included in the drivers are:

```
        db      'SEQ'                   ; marker for command sequence tables
seqtbl2p:
        dw      seqtbl2                 ; address of table-2
seqtbl1:; table for the initial characters of command sequences
        db      ESC                           ; <ESC ... >
        db      ^CMDSEQ|^CSBEGIN              ; may be a 2, 3, or n-character seq

        db      DC2                           ; <DC2 n>
        db      ^CMDSEQ|^CSBEGIN|^CSNXLAST;   2-character sequence

        db      80H                           ; end of table

seqtbl2:; table for the second characters of command sequences
        db      ';'                     ; <ESC ;>
        db      ^CMDSEQ|^CSLAST         ; this is the last char in the sequence

        db      ':'                     ; <ESC : n>
        db      ^CMDSEQ|^CSNXLAST       ; this is the next-to-last char in seq
        db      '1'                     ; <ESC 1 nl ... nk NULL>
        db      ^CMDSEQ                 ; NULL terminated sequence
        db      '2'                     ; <ESC 2 n>
        db      ^CMDSEQ|^CSNXLAST       ; this is the next-to-last char in seq

        db      '3'                     ; <ESC 3 nl ... nk NULL>
        db      ^CMDSEQ                 ; NULL TERMINATED sequence

        db      '4'                     ; <ESC 4>
        db      ^CMDSEQ|^CSLAST         ; this is the last char in the sequence

        db      '5'                     ; <ESC 5>
        db      ^CMDSEQ|^CSLAST         ; this is the last char in the sequence

        db      '6'                     ; <ESC 6>
        db      ^CMDSEQ|^CSLAST         ; this is the last char in the sequence

        db      '7'                     ; <ESC 7>
        db      ^CMDSEQ|^CSLAST         ; this is the last char in the sequence

        db      '8'                     ; <ESC 8 n>
        db      ^CMDSEQ|^CSNXLAST       ; this is the next-to-last char in seq

        db      '9'                     ; <ESC 9 n>
        db      ^CMDSEQ|^CSNXLAST       ; this is the next-to-last char in seq

        db      80H                     ; end of table
```

USING A TAPE DRIVE

The tape driver may reside in IOP1, IOP2, IOP3, or IOP4.

When the system is booted, an IOP is loaded by **/etc/iostartup.cmd** with **/dev/iop/tape.iop**. This file includes the tape driver, TP, and the IOP memory driver, IOMEM.

Since the tape driver includes an 8K buffer, there is not enough room for the IOP to include TP as well as the QTTY and QSLPT drivers. (QTTY, QSLPT, and IOMEM are in **/dev/iop/cromix.iop**).

The parameters of TP can be set with the **.setmode** system call. These parameters are defined in the file **/etc/tmodequ.z80**. The Mode utility can also be used to set them. Type **help mode** for further information on TP devices.

The **Ddump** utility is often used with TP. Type **help ddump** or turn to the **Ddump** utility in Chapter 9 of this manual for information on how to use it.

SETTING UP A MULTI-USER SYSTEM

Multi-user configurations are available with two kinds of I/O processing. TU-ART device drivers are used on all single user and some multi-user systems. The alternative systems use IOP/Quadart devices. An IOP system contains a separate microprocessor for handling input and output.

The Cromix diskette you received is set up for a single user system. It assumes that the main terminal is connected to the serial port on the 16FDC board and that the system printer is a Cromemco 3703 dot matrix printer. To set up the system for more than one user, follow this procedure:

1. Set up your hardware in the desired arrangement. Refer to Appendix A for the correct switch settings for different circuit boards in the system. The main terminal must be connected to the serial port on the 16FDC board.

2. Boot up the system on the new diskette.

3. Make the appropriate entries in the file **/etc/ttys** so that the system software is aware of the other terminals connected to the system. A detailed explanation of this process is found below in **Changing Entries in the Ttys File.**

   The **ttys** file is consulted only when the system is booted or when the **kill -1 1** command is given. Until one of these events occurs, changes to this file have no effect.

4. If the additional terminals are connected to a TU-ART board, give the following command to incorporate these changes:

   kill -1 1

5. If the additional terminals are connected to a Quadart board, or a combination of TU-ART and Quadart boards, see the section titled **Incorporating Changes for a Quadart Board.**

6. Use the **Passwd** utility described in Chapter 9 for establishing new users.

Changing Entries in the ttys File

The **/etc/ttys** file contains information about which terminals are connected to the system. There must be a **1** in the first column of an entry in the **ttys** file for each terminal which is to be online.

Example:

```
1:n:tty1
1:a:tty2
1:a:tty3
0:a:tty4
0:a:tty5
0:a:tty6
```

In this example, tty1, tty2, and tty3 display the login message. Any other terminals attached to the system do not receive the login message and are ignored.

By editing the **ttys** file, you may alter the configuration of your system software. To use the Screen Editor to modify the **ttys** file, enter:

screen /etc/ttys RETURN

This will cause the following to be displayed on the terminal:

```
1:n  tty1
0:a  tty2
0:a  tty3
      .
      .
      .
0:a  qtty1
0:a  qtty2
      .
      .
      .
0:a  mtty1
0:a  mtty2
      .
      .
      .
```

If there is a second terminal to be connected to the TU-ART board, use the **Xchng** command in the Screen Editor to replace **0** with **1** in the line:

```
0:a  tty2
```

If you need to connect a terminal to a Quadart board, replace **0** with a **1** in the line:

```
0:a  qtty1
```

Follow the appropriate procedure for each additional terminal in your system. The mtty lines are used for modems connected to a Quadart board.

Also note that **n** in the line

```
1:n  tty1
```

represents **no change** and needs to be replaced with either an **a** to mean **auto baud** or the actual baud rate of the terminal connected to the 16FDC board. (The main terminal should be set for a baud rate of 9600).

For example, in a four user system with the second, third, and fourth users connected to a Quadart board, the table appears as follows:

```
1:9600    tty1
0:a       tty2
          .
          .
          .
1:a       qtty1
1:a       qtty2
1:a       qtty3
          .
          .
          .
```

The devices selected in the **/etc/ttys** file need not be in numerical sequence. Devices are chosen based on port availability in the system. Consult the device definition tables in the back of this manual for the appropriate device name of a particular port address. Thus it is possible to use

```
1:n  tty1
1:a  tty4
1:a  tty6
```

if these were the only serial ports available in the system. Obviously, it is clearer to use the devices in order wherever possible.

The changes to the **ttys** file will have no effect until the system is rebooted or until the following command is given:

 # kill -1 1

SETTING UP AN IOP/QUADART BASED SYSTEM

To set up a Cromix system with terminals connected through Quadart boards, you must reconfigure the software and then make the necessary alterations in the hardware. The procedure for each process is explained in the following sections.

Setting Up the Software with Quadarts and TU-ARTs

The first step is to execute the **Runqd** (Run Quadart) utility. **Runqd** is a command file which makes several necessary changes in the root directory.

If you are planning to use more than one terminal with your system, you must now modify the **/etc/ttys** file, as explained previously in the section titled **Changing Entries in the ttys File.** **Runqd** configures **/etc/ttys** for the system console only.

Now, if your root disk is not the same as your boot disk, use the procedure shown in Steps 1 through 4 below.

1. Execute **/gen/default**, specifying the correct default root device for your system. This is usually the hard disk. For example,

   # /gen/default /cromix.sys 2 0

2. Mount the boot disk using the **Mount** utility described in Chapter 9 of this manual.

3. Using the **Copy** utility, copy **/cromix.sys** to the boot disk.

4. Using the **Wboot** utility, write the boot track to the boot disk.

5. Now unmount the boot disk, using the **Unmount** utility.

If you are using more than one IOP for terminals or serial devices such as printers, modify **/etc/iostartup.cmd** by using the Screen Editor to delete the **%** before the lines for **iop2**, **iop3**, or **iop4**, as necessary. The **%** indicates a comment which is not executed.

Setting Up the Hardware

Caution: The system must be turned off and unplugged before you make changes to the hardware. The first step is to adjust the switch settings on the IOP and Quadart as shown in the diagrams in Appendix B of this manual. For best results, you should be using version 03.00 or higher of the IOP monitor.

If you are connecting more than one Quadart to your IOP, modify the priority header for each Quadart after the first, according to the diagram in Appendix B. For the first Quadart connected to your IOP, the factory setting of the priority header is correct.

Now install the IOP and Quadarts in the S-100 bus. Install the C-bus cable.

Do not connect the priority jumper cable to the Quadarts, as Quadarts are not in the S-100 priority interrupt chain.

However, the third and fourth Quadarts connected to any one IOP must be connected with a separate priority interrupt jumper cable. This jumper cable must not be connected to any other boards.

Now connect each IOP to the S-100 interrupt chain. The connection should run from the OUT connection on the 4FDC or 16FDC to the IOPs and TU-ARTs, and then to the PRI board.

The internal signal cable for the system console should be connected to the 4FDC or 16FDC. The internal signal cables used to connect local terminals or serial devices (such as printers) should be connected to one of the pin sets on the outside of the Quadarts. These are connectors J2, J4, J6, and J8.

The cables used to connect modems or a 3102 terminal from the AUX port must be connected to one of the inside pin sets. These are connectors J1, J3, J5, and J7.

The 16FDC switches 2 and 3 must be ON. If no terminal is connected to the 16FDC, switch 5 must be ON. Terminal baud rates up to 19200 baud are permissible, except for the system console which must be set to 9600 baud or slower.

Setting Up the Software with Quadarts Only

Running a system with both Quadarts and TU-ARTs requires that both the Quadart and TU-ART drivers be present. This uses five more system buffers than are used by the Quadart driver alone. This section describes the procedure used to bring up a system using Quadart drivers only.

To set up a system without TU-ART drivers (i.e., no terminal connected to the 16FDC), follow these steps.

1. Link the console to the **qttyl** device by giving the command

   # maklink -f /dev/qttyl /dev/console

2. Run **Crogen,** answering NO to the first question and providing a default root device.

3. Set switches 2, 3, and 5 on the 16FDC to ON.

4. Deselect **ttyl** in the **/etc/ttys** file.

5. Boot up using the system file you generated in step 2.

Booting the IOP/Quadart System

Your hardware and software are now modified to accommodate an IOP/Quadart based system. You are now ready to boot up. Insert the boot disk in the appropriate drive and proceed as you would for a TU-ART based Cromix system. The boot process takes just a bit longer when using IOP/Quadart terminals than it does for a TU-ART or single user configuration.

When **startup.cmd** finishes execution, the login message appears on all connected terminals.

CONNECTING A REMOTE TERMINAL THROUGH THE MTTY DEVICE

The mtty driver is designed to connect remote terminals over phone lines via modems. This is a dial-in line only. Use the qtty driver for outbound communications. The mtty driver has these advantages over the standard tty and qtty drivers:

1. When the user hangs up (breaks the connection) without logging off, the Cromix system sends a SIGHANGUP signal to all processes started by that user if the SIGHUPALL mode bit is set. This includes all detached processes. Note that the system interprets loss of Data Carrier Detect and Clear To Send as disconnection.

2. When the user logs off, the Cromix system drops DTR for a short period and then raises it again if the SIGHANGUP mode bit is set. This hangs the modem up permiting another user to phone in and log on.

To connect a remote terminal to an mtty port:

1. The Cromix system must include drivers for terminals connected to Quadarts (see **Crogen**).

2. A device file for the mtty should be set up in the **/dev** directory using **Makdev.** The major device number is 2 and the minor device number is the number of the qtty device for the port connected, plus 128. For example, if you wish to connect the remote line to the 4th port on an existing Quadart (assuming that this is the first Quadart on the system), the normal minor device number is 3. Adding 128 to 3 gives you a minor device number of 131. The Cromix Operating System disks come with mtty1 through 4 already in the **/dev** directory.

3. The proper entry in the **/etc/ttys** file should be made. If modems of differing baud rates are expected, use autobaud.

4. The connection from the Quadart to the system back panel is the same as any other for serial port. A special 12-wire cable is required from the back panel to the modem. The Cromemco part number is 519-0117. Refer to the Cromemco MDM-1200 manual for more information.

5. Any asynchronous modem may be used (baud rate is limited by the modem), as long as modems on both ends are compatible. Some modems have the option to strap DTR (data terminal ready) high. DO NOT use this option on the modem on the computer end, because the driver manipulates DTR.

ADJUSTING THE SYSTEM CLOCK

The system clock may be adjusted if it consistently runs fast or slow. This may be accomplished by a privileged user changing the real time clock driver (timer) by using the **Mode** utility.

The 16FDC board should be modified to incorporate the real time clock interrupt feature for maximum system clock accuracy. This modification is made at the factory for all 16FDC rev F1 mod level 3 or higher 16FDC boards. Refer to the 16FDC Manual for information on this modification.

The following command combined with the response from the operating system displays the correction factor in use.

```
# mode timer

Character Device 4:0
Correction (seconds/100 days) 3800
```

To alter this, give the following command:

```
# mode timer c xxxx
```

In the command above, **xxxx** is the new correction factor in the range ±32767. This command should be included in the **/etc/startup.cmd** file so that the new correction factor is automatically set each time the system is booted.

The correction factor can be calculated by performing the following steps.

1. Determine the deviation of the system clock in seconds per 100 days. Assume, for example, that the system clock gains 1 minute each day. A gain of 1 minute per day is the same as 60 seconds per day or 6000 seconds per 100 days.

2. Compute the new correction factor. Add (or subtract) the deviation to (or from) the current correction factor. In this example, the current correction factor is 3800 and the deviation is 6000. This yields a new correction factor of 9800.

Chapter 7

THE CROMIX SHELL

This chapter explains the operation of the Cromix Shell. The Shell is the program that interprets and processes all commands.

The Shell insures that arguments typed on the command line are available for use by the called programs. It also allows more than one command to be entered on the command line (sequential and concurrent processes). The Shell sends output to a file and accepts input from a file (redirected I/O).

The Shell handles all file and device dependent information. All directories are created, changed, and displayed by using Shell commands.

In this manual, the term **command** refers to Shell commands intrinsic to the Cromix Operating System. The term **utility** refers to utility programs stored on the disk. A command executes within the system bank of memory; a utility requires additional memory for execution.

The Cromix Shell uses three standard files. These are the standard input file, (**stdin**), the standard output file, (**stdout**), and the standard error file, (**stderr**). As shipped, all three refer to the console; standard Shell input is from the console keyboard and standard output and error messages to the Shell go to the console screen. Unless otherwise stated, assume that stdout, stdin, and stderr all refer to the terminal.

COMMAND SYNTAX

Each Cromix Shell command has the following syntax:

```
filename -options arg1 arg2 ...
```

where **filename** is the name of a file, **-options** are optional options, and **arg1**, **arg2**, and so on are optional arguments. The Shell program searches for **filename** as follows:

1. filename.bin (in the current directory)
2. filename.com (in the current directory)
3. filename.cmd (in the current directory)

4. /bin/filename.bin (in the /bin directory)
5. /bin/filename.com (in the /bin directory)
6. /cmd/filename.cmd (in the /cmd directory)

If the file is not found in these directories, the system displays an error message. If the file is found, it is treated according to the file naming conventions outlined in Chapter 4. A file with the extension **.cmd** is assumed to have the command **Shell** at the beginning of the command line.

SEQUENTIAL AND DETACHED PROCESSES

More than one Shell command may appear on a single command line. A command followed by a semicolon (;) means that any command that follows is executed after the first command has finished execution. This is called **sequential processing.**

A command followed by an ampersand (&) means the process specified by the command is executed as a **detached process** and that any subsequent command on the line is executed as a **concurrent process.**

When a detached process begins execution, a process identification number (**PID**) is displayed on the terminal. For <u>each</u> additional detached process executed concurrently, one additional bank of memory usually is required. If there is not enough memory in the system, the system displays an error message. (There are exceptions. Some processes use only part of a bank of memory and some do not use user memory but run in the system bank.)

For example, if **a** and **b** are commands, each of which begins execution of a single process, then:

```
% a;b
```

causes process **a** to begin and complete execution before process **b** begins execution - sequential processing. And:

% a&b

causes process **a** to begin execution in the detached mode and process **b** to begin execution at the same time (concurrent processing).

If a single command is given on a line terminated with an ampersand (&), the process specified by the command begins execution in the detached mode, a PID number is assigned and displayed, and the Shell immediately prompts for another command.

Executing the Shell command **Wait** suspends execution of any additional commands until all detached processes have finished execution.

REDIRECTED OUTPUT

Output that normally goes to the standard output device (the terminal) may be redirected to a file. This file may be an ordinary file, a device, or a program. Output can be redirected by entering a greater-than sign (>) followed by the output file or device name on the command line.

% ty novel.txt > newnovel

This command types out the contents of the file **novel.txt**, but redirects the output to the file **newnovel**, rather than sending it to the console. If the new file does not exist, it is created. If the file already exists, the contents of the file are overwritten. Care must be exercised when using redirected output to avoid deleting files accidentally. The following is an example of this type of mistake:

% ty novel.txt > novel.txt

After this command is executed, the file is empty. A redirected file always deletes the contents of the receiving file as its first operation.

A double greater-than sign (>>) appends the output to the specified file. If the file already exists, the output is placed at the end of the existing file without overwriting the contents of the file. If the file does

not exist, a new file is created in the same manner as the redirected file. For example:

```
% ty new_notes >> notes
```

The command line above appends the file **new_notes** to the file **notes.**

REDIRECTED ERROR MESSAGES

Output normally goes to the standard output file (the terminal). However, when output is redirected with either > or >>, output goes to the specified file. Errors are not redirected; they are sent to the standard error file, which is usually the console. To redirect error messages, an asterisk (*) must follow the redirected (>) or appended (>>) output. Redirecting error message output can be useful in a number of applications, preventing display of the output from a background job from appearing on the console. For instance, if a background job is run simultaneously with the Screen Editor, error messages from the job could disrupt the edit. The following command line shows how to redirect error messages from the standard output to a file:

```
% ty a >* b
```

This command line sends the contents of file **a** to file **b**, along with any error messages generated during the process. If **b** already exists, its contents are overwritten, as though output had been redirected without using an asterisk.

```
% ty a >>* b
```

The command line above, like the one before it, redirects error messages routed to the standard error device. In this case, the new data is appended to the end of file **b**, if **b** already exists. If **b** does not exist, it is created. The error messages are also redirected.

REDIRECTED INPUT

Input may be redirected so that it comes from a file rather than from the standard input device. To redirect input, enter a less-than sign (<) followed by the name of the input file or device. For example:

```
% proc < infile
```

REDIRECTED INPUT AND OUTPUT

Both input and output can be redirected on the same command line. For example:

```
% screen notes < infile > /dev/null
```

In this example, the Screen Editor is called to edit the file **notes**, using edit commands from **infile.** The output from the editor is redirected to a null device rather than being displayed on the console or being saved in a file.

PIPES

A **pipe** connects the output of one program to the input of another, so the processes run in sequence, forming a pipeline. Pipes are often used in place of input and output redirection.

```
% l > temp ; spool temp; del temp
```

This command line lists the contents of the directory, sends it to the printer via the **Spool** utility, and deletes the temporary file. The same result can be achieved using a pipe. For example:

```
% l | spool
```

The vertical bar (|) between these two commands is the pipe symbol. It directs the output of the process on the left into the process on the right. When pipes are used, the processes run concurrently. This requires

more memory than running them in sequence. The **sequential pipe** (><) (less-than sign, greater-than sign) is used when memory resources are at a premium. The redirected sequential pipe executes the first command, saves its output in a temporary file, and uses the contents of that temporary file as an input to the second process.

% 1 >< spool

% 1 | spool

Both command lines above list the directory and send the output to the Spool program. The command line using the sequential pipe takes less memory and more time than the command line using the standard pipe.

Any program that reads from the console can read from a pipe instead, and any program that sends output to the terminal can also drive a pipe. An asterisk (*) immediately after a pipe or a sequential pipe redirects the standard error output and the standard output. If the command:

% command1 ><* command2
% command1 |* command2

is given, the standard output and the standard error output are piped into **command2.**

TEES

It may be desirable to pipe the output of a process to another process and to the standard output simultaneously. To do this, the **Tee** command is placed after the pipe symbol on the command line. For example:

% command1 | tee command2

The command line **Tee** pipes the output of **command1** to the input of **command2** and, in addition, sends the output to the console. **Tee** may also be used with the **sequential pipe**, producing the same effect as the standard pipe but taking longer. The output that **Tee** sends to the console can be redirected again using simple or appended file

symbols. The following example demonstrates this function:

```
% l ><* tee file1 > >< tee file2
% l |* tee file1 > | tee file2
```

Both command lines list the directory and send the output to **file1**, **file2**, and the terminal. Using standard pipes takes less time than using sequential pipes, but standard pipes require more memory. Note that a file or a command may directly follow a tee, but a standard redirection symbol (>) must be included with a pipe or sequential pipe to allow the piped output to go directly to a file.

IMPORTANT NOTE

Even though a command such as

```
% ty novel.txt > /dev/prt
```

sends the contents of the file **novel.txt** to the printer, it is not a good idea to redirect output to the printer on multi-user systems. The reason is that, if two or more users or processes attempt this operation simultaneously, the results are unpredictable. For access to the printer, use the **Spool** utility instead.

PARENTHESES ON THE COMMAND LINE

Parentheses are used to group commands on the command line. They can be used to send output from several sequential processes to the same file. For example:

```
% (a;b) > xyz
```

The same output file results from the command:

```
% a > xyz ; b >> xyz
```

The command line sends output from process **a** to file **xyz** and appends the output from process **b** to the same file.

Parentheses can also make two or more sets of sequential processes execute simultaneously as detached processes. The following command line executes processes **a**, **b**, and **c** in one bank of memory, while processes **d**, **e**, and **f** are executed in another:

```
% (a;b;c) & (d;e;f)
```

The command line can be terminated with an ampersand (&) to cause execution of both processes in detached mode, while the Shell prompts the user for another command.

QUOTATION MARKS ON THE COMMAND LINE

Pairs of quotation marks (") or apostrophes (') may be used to enclose strings of special characters on the command line. For example, the following command line displays a greater-than sign within a message:

```
% echo "this is a special character:  > right"
this is a special character:  > right
```

If the quotation marks were omitted, the output would be redirected to the file named **right**.

Quotation marks are used to pass the special characters representing ambiguous file references (*, **, and ?) as arguments to programs. Enclosed in quotation marks, these characters lose their special significance and are passed by the Shell without expansion. Please refer to Chapter 4 for additional information.

ARGUMENT SUBSTITUTION

Arguments from the command line are substituted in sequential order in a command (**cmd**) file for each appearance of #1, #2, #3, . . ., #9. Assume that the command file named **test.cmd** contains:

```
% ty #2 #1
```

If the command:

% test file_x file_y

is given, the first argument, **file_x**, is substituted for **#1** in the command file and the second argument, **file_y**, is substituted for **#2**. The result is the same as if the command:

% type file_y file_x

had been entered.

As many arguments as are entered on the command line may be substituted by the use of **#***. For example, the following line in a command file displays all arguments with which the command file was called.

```
echo #*
```

WRITING COMMAND FILES

Any command or sequence of commands that can be entered on the command line can also be put in a file and executed by entering the name of that file on the command line. A file that contains one or more commands to the operating system is called a **command file**.

A command filename must have the extension **.cmd** and must reside in the current directory, or the **/cmd** directory, to be found automatically by the Cromix Operating System. Once the command file is written, it is executed by entering the name of the file, less the **.cmd** extension, on the command line.

The ability to add user defined commands to the Cromix Operating System gives the user the power to customize the system for virtually any application.

Parameters may be passed to any command file from the command line by referring to those parameters in the command file by a **#**, followed by a number. The number refers to the specific parameter on the command line.

In a command file, jumps and conditional jumps can be made into labels by using the **Goto** and **If** Shell commands. Using the **Echo** utility, command files send output to the standard output. The following example is a listing of the command file **echo_args.cmd** and illustrates the use of some of these Shell commands:

```
%start
if .#1 = .  goto done
echo #1
shift
goto start
%done
```

The following was typed on the command line to invoke execution of the command file **echo_args.cmd.**

% **echo_args one two**

This command invokes execution of the command file **echo_args.cmd** in which argument **#1** is **one** and argument **#2** is **two.**

The first line of the command file, **%start,** acts only as a label.

The second line uses the **If** Shell command to test if the string produced by joining the argument **#1** to the end of the character . equals the string . (period). Since the string **.#1** expands to **.one,** it does not equal the . string. The condition is false and control passes to the next line.

If no arguments had been given on the command line that called **echo_args,** then argument **.#1** would have expanded to . (the period by itself). The test

if . = . goto done

would then be true, and control would be passed to the line labelled **done.**

The next line expands into **echo one,** sending the string **one** to the console on a line by itself.

The next line is now executed. It is the **Shift** command. **Shift** moves all the arguments in the argument list to the left one place. Argument **#1** is now the string **two.** The statement that follows is a **Goto** to the line labelled **start.** It repeats the sequence described above, except that the string **two** is printed this time through the loop. The argument list is again shifted by one and control jumps to the beginning. This time, the **If** command transfers control to the line labelled **done.** Execution of the command file terminates and control returns to the operating system.

SHELL COMMANDS

This section describes the **If**, **Kill**, **Path**, and **Sleep** Shell commands.

If

The **If** Shell command allows the programmer to write command files that execute commands conditionally. For example:

```
command1
If -err command2
.
.
.
```

In this example, **command2** is executed if **command1** returns an error code when it terminates. Otherwise, command2 is not executed, and execution continues with the subsequent line.

Kill

The **Kill** Shell command sends a specified signal to a specific process. If the signal type is not specified, **Kill** defaults to a terminate signal. For example,

Kill -2 1	kills all processes and shuts the system down, and
Kill 0	aborts all background jobs initiated from the user's terminal.

Path

The **Path** Shell command finds the directory location of an executable file or command file. If the specified command is a Shell command, **Path** so notifies the user.

Sleep

The **Sleep** Shell command suspends execution for the number of seconds specified by the argument.

COMMAND LINE LENGTH

Each individual Shell command line is limited to 512 characters after ambiguous references have been expanded.

SYSTEM BUFFERS

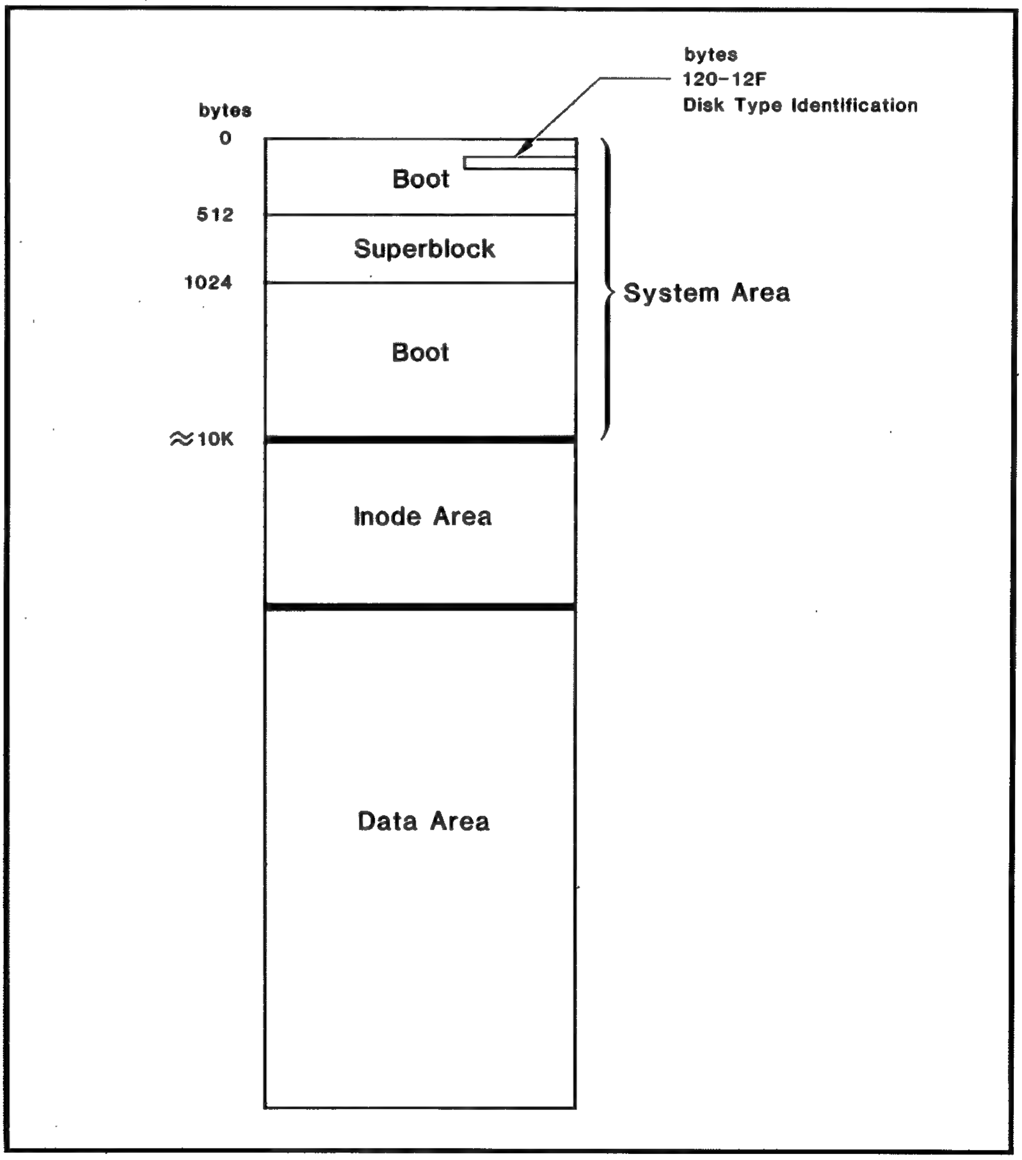
When the Shell evaluates a command line, it is placed in a system buffer. The number and size of system buffers vary according to the size of the Cromix Operating System. If there are insufficient system buffers to evaluate a command line, the error message, **no system buffers available**, is displayed and the command line is not executed.

Because system buffers are released after a command line is executed, the error condition is usually temporary and the command can be reentered and executed.

Chapter 8

DISK ALLOCATION UNDER THE CROMIX OPERATING SYSTEM

This chapter describes disk allocation under the Cromix Operating System. Any small or large floppy disk or hard disk formatted for use under the Cromix system is divided into three major sections: the **System Area, Inode Area,** and **Data Area.** These disks are formatted with a block size of 512 bytes decimal.



Layout of a Cromix Disk

SYSTEM AREA

The System Area has a default size of 10K bytes for all disk types. Although it is not recommended, the size of this area can be specified when running the **Makfs** (make file system) utility program.

The System Area contains system information required for booting up (boot tracks) and disk type identification. In addition, it contains the Superblock, and, for hard disks, the alternate track table.

Boot Up Information

The entire System Area of a disk is dedicated to the system information required for booting the system, with the exception of the disk type identification area, the Superblock, and, on hard disks, the alternate track table.

Disk Type Identification

On floppy disks, bytes 120 through 127 (in the first block) contain ASCII encoded data detailing the type and use of the disk.

Floppy disks have six letters in this position. When formatted for use with the Cromix Operating System, byte 120 contains a **C.** Byte 121 contains an **S** or **L**, to indicate a Small (5") or Large (8") floppy disk. Bytes 122-123 contain the characters **SS** or **DS**, indicating a Single Sided or Double Sided Disk. Bytes 124-125 contain the characters **SD** or **DD**, indicating a Single Density or Double Density disk. Bytes 126-127 are not significant, but are reserved for future use.

On hard disks, bytes 68h through 7Fh contain disk type identification. The following table details this area of the disk. Older 8" hard disks contain **CH11SD** in the disk identifier field. Although Cromemco software will support this designation, new disks will be identified as **C8-1** (8" hard disk) and **C5-1** (5" hard disk).

68-69	Number of cylinders (2 bytes)
6A-6B	Number of alternate tracks (2 bytes)
6C	Number of surfaces (1 byte)
6D	Number of sectors per track (1 byte)
6E-6F	Number of bytes per sector (2 bytes)
70-71	Byte count of start of alternate track table (2 bytes)
72-73	Cylinder number of start of disk (2 bytes)
74-77	Reserved for future use (4 bytes)
78-7B	Hard disk identifier (4 bytes)
7C-7F	Reserved for future use (4 bytes)

The following structure will be added to cdosequ.z80 for use by software which will be accessing these values.

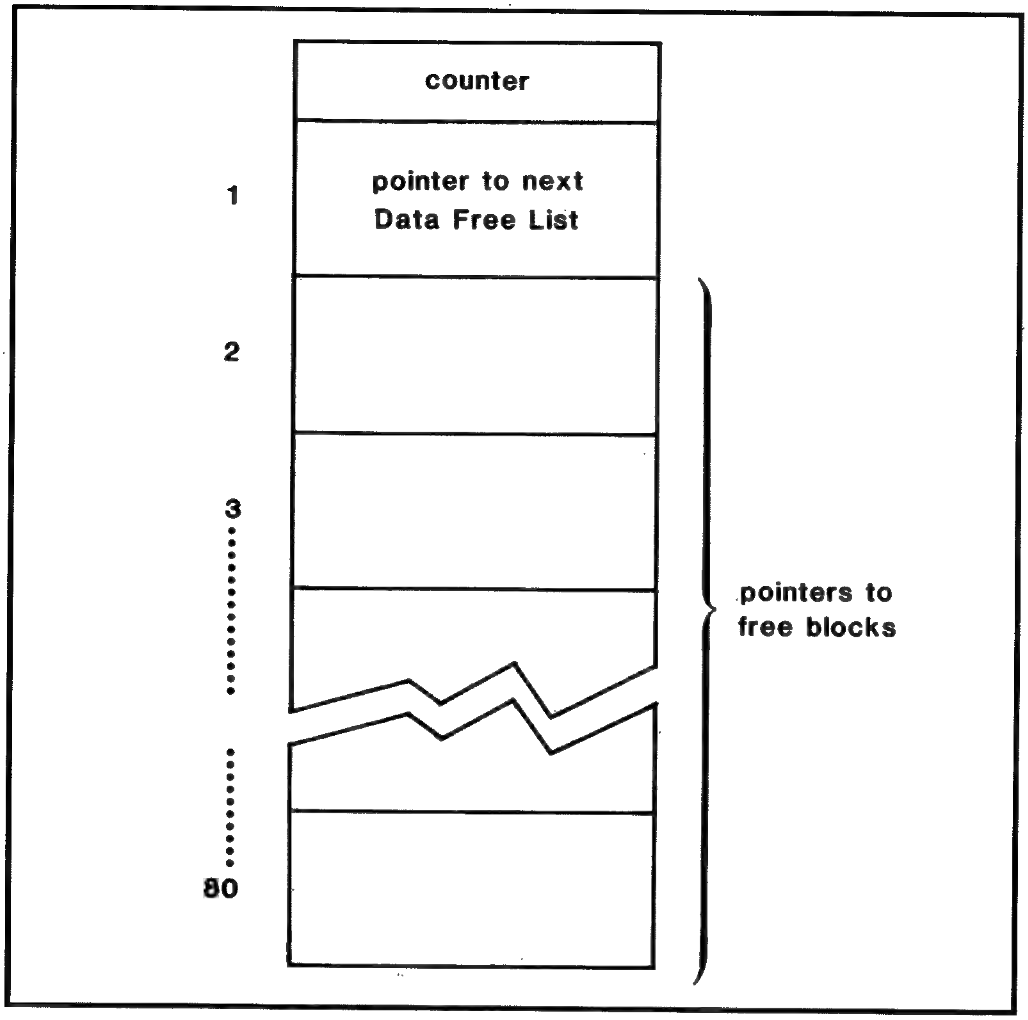
;* Structure for disk type specifier for hard disks

```
        struct  68H
dskcyl  defs    2       ; Number of cylinders (not
                          including alts.)
dskanm  defs    2       ; Number of alternate
                          tracks
dsksur  defs    1       ; Number of surfaces
dsksct  defs    1       ; Number of sectors/track
dskbsc  defs    2       ; Number of bytes/sector
dskatt  defs    2       ; Byte count of start of
                          alternate track table
dskdsk  defs    2       ; Cylinder number of start
                          of disk
        defs    4       ; Reserved for future use
dskid   defs    4       ; Hard disk identifier
        defs    4       ; Reserved for future use
        mend    struct
```

Superblock

The second block (bytes 512-1023) is the Superblock. This block contains housekeeping information for the disk, including the **Block Free List** and the **Inode Free List.**

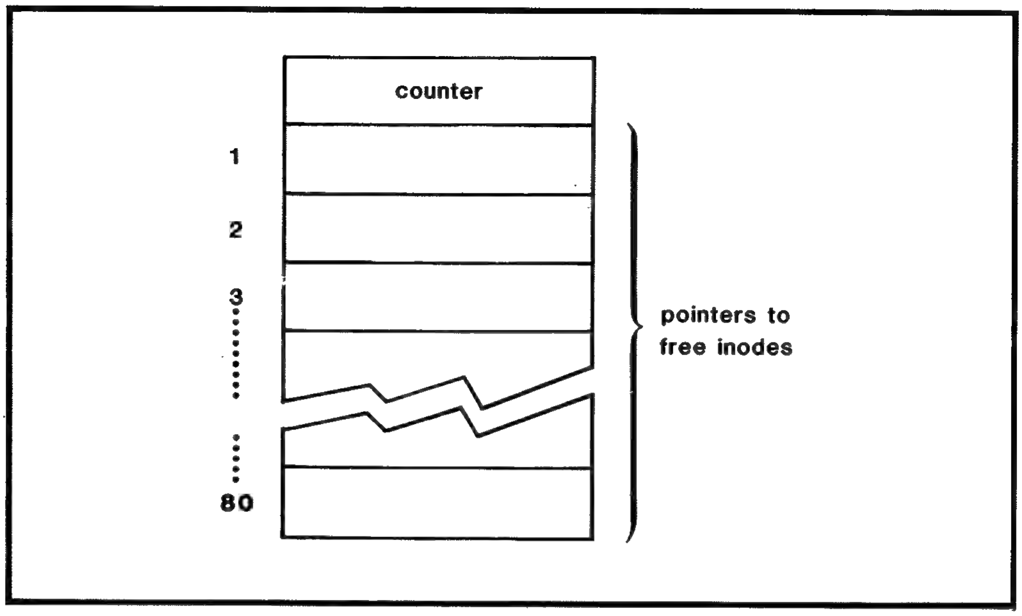
The Block Free List (sometimes called the Free List) is a stack of 80 4-byte pointers, preceded by a 2-byte counter. Each pointer in the Block Free List points to a disk block not in use. As information is deleted from the disk, the Block Free List grows; as information is written to the disk, it shrinks.



Block Free List

The last pointer used (actually, the first pointer in the list) points to a block on the disk that contains another Block Free List. When the Block Free List in the Superblock is exhausted, the next Block Free List is loaded into the Superblock. When the Block Free List in the Superblock is full, it is moved to the Data Area of the disk.

The Inode Free List is a stack of 80 2-byte inode numbers preceded by a 2-byte counter. Each entry in the Inode Free List is the number of an unused inode. When this stack is exhausted, the Cromix Operating System searches through the inode table and replenishes the stack with the numbers of additional inodes not in use.



Inode Free List

Alternate Track Table

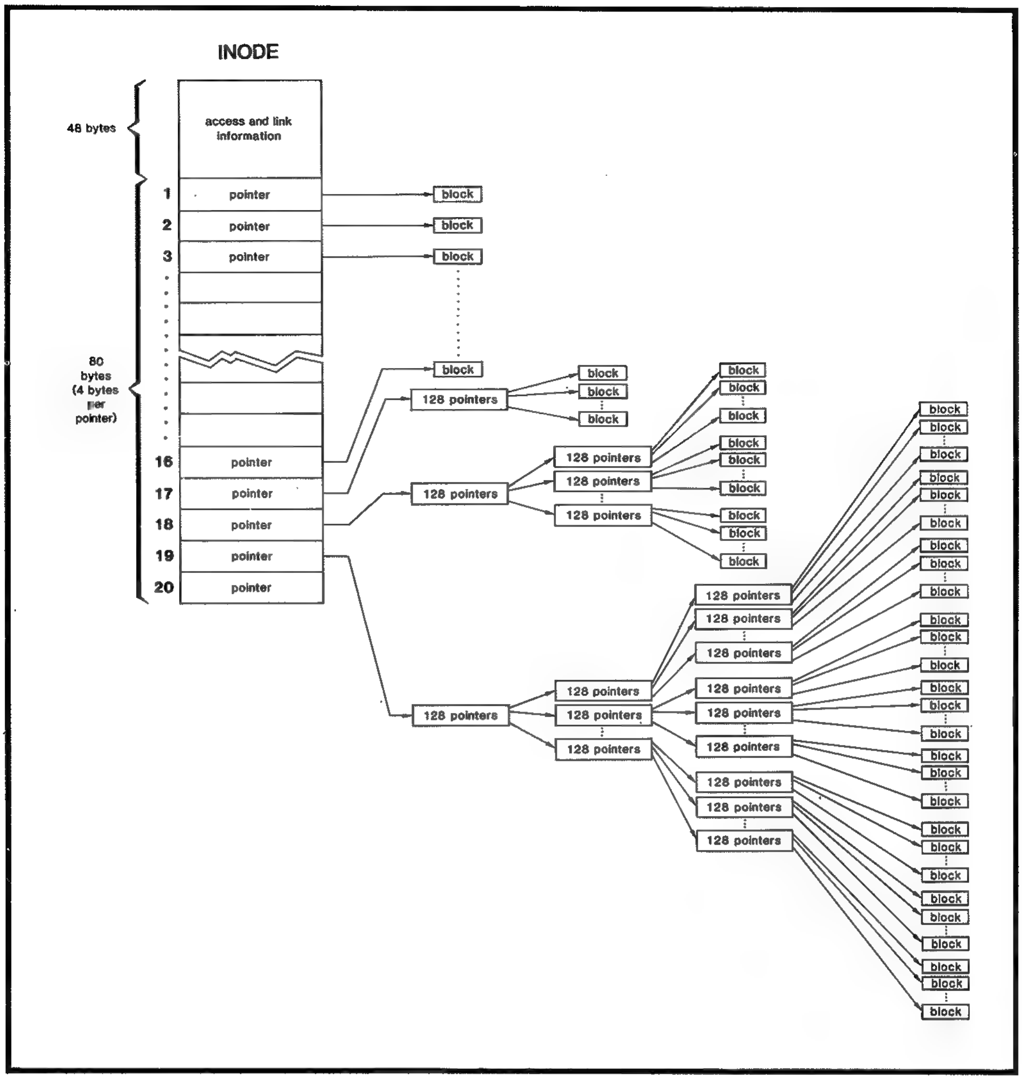
The Alternate Track Table for the hard disk is located at the top of the System Area, before the Inode Area.

INODE AREA

An inode is a descriptor for one file; it contains a collection of information pertaining to the file.

The first 48 bytes contain information on the number of links to the file, allowable access modes, and most recent access times for various types of access.

The last 80 bytes of the inode contain 4-byte pointers to the file itself.



Inode Layout

The first 16 of these pointers each points to a block of the file. The first pointer points to the first block (bytes 0-511); the second pointer points to the second block (bytes 512-1023), and so on. This continues until the whole file has been pointed to, or until the sixteenth pointer has been used (pointing to bytes 7680-8191). Thus, if the file is 8 Kbytes or smaller, only the first 16 (or fewer) pointers need be used.

If the file described by the inode is larger than 8 Kbytes, the seventeenth pointer is used. This pointer points to a block of 128 pointers. Each of these pointers points to a block of the file in a manner

similar to the first 16 pointers described above. Thus the seventeenth pointer describes the next 64 Kbytes of the file.

If the file is larger than 72 Kbytes, the eighteenth pointer is used. This pointer points to a block of 128 pointers. Each of these points to a block of 128 pointers. These pointers, in turn, point to a block in the file. Thus, the eighteenth pointer describes the next 8192 Kbytes of the file.

The nineteenth pointer extends one more level, covering the next 1,048,576 Kbytes of the file.

Because the first 19 pointers can describe a file of over a gigabyte (one billion bytes or characters), a twentieth pointer has not yet been implemented.

DATA AREA

The Data Area occupies most of the disk. All data on the disk is stored in the data area. All blocks pointed to by inodes are in this area.

INODE, BLOCK, TRACK, AND CYLINDER NUMBERS

The following discussion and formulae pertain only to the Cromemco **11 Mbyte Hard Disk Drive.**

Definitions

There are three **surfaces** on the hard disk to which data is written. There are 350 concentric **cylinders,** and each cylinder includes three surfaces. The intersection of a surface and a cylinder is a **track.** A track is composed of 20 **sectors.** To summarize:

There are 3 surfaces per cylinder.
There are 20 sectors per track.
There are 60 sectors per cylinder.

Converting Inode Numbers to Block Numbers:

$$\frac{\text{inode} - 1}{4} + 20 = \text{block}$$

Converting Block Numbers to Inode Numbers

inode = (block-20) * 4 + Z

where Z = 1, 2, 3, or 4

Converting Block Numbers to Cylinder, Surface, and Sector Numbers:

$$\frac{\text{block}}{\text{sectors per track}} = \text{quotient \& remainder}$$

The above division yields a quotient (whole number) and a remainder. The remainder is the **logical sector number.** Use the following table to convert the logical sector number to the **physical sector number.** The quotient is used in the following division to obtain the cylinder and surface numbers:

$$\frac{\text{quotient}}{\text{surfaces per cylinder}} = \text{quotient2 \& remainder2}$$

The above division yields quotient2, the **cylinder number** and remainder2, the **surface number.**

Converting Logical and Physical Sector Numbers:

------Sector------ Logical	Physical	------Sector------ Physical	Logical
0	0	0	0
1	7	1	3
2	14	2	6
3	1	3	9
4	8	4	12
5	15	5	15
6	2	6	18
7	9	7	1
8	16	8	4
9	3	9	7
10	10	10	10
11	17	11	13
12	4	12	16
13	11	13	19
14	18	14	2
15	5	15	5
16	12	16	8
17	19	17	11
18	6	18	14
19	13	19	17

Converting Cylinder, Surface, and Sector Numbers to Block Numbers:

First the physical sector number is converted to a logical sector number using the table. Then the following formula yields the block number:

((cylinder * surfaces per cylinder) + surface) *

sectors per track + logical sector = block

Chapter 9

SHELL COMMANDS AND UTILITY PROGRAMS

The Cromix utility programs perform many necessary functions. They are similar to and used in conjunction with the Cromix Shell commands.

In contrast to the Shell commands, utility programs are not intrinsic to the Cromix Operating System but must be called from disk when needed. While Shell commands require only the system memory bank for execution, utility programs use additional memory.

Write and append access for all utilities is limited to privileged users.

The following list summarizes the commands and programs described in detail in this chapter.

SUMMARY OF COMMANDS AND UTILITIES

access	changes access privileges of a file
backup	backs up a directory and its descendant files
blink	links files together; used with the Crogen utility
boot	loads an operating system into memory and begins execution
check	runs the dcheck and icheck utilities.
cdoscopy	copies files to and from a CDOS disk
chowner	changes the owner or group owner of a file
cmpasc	compares 2 text files
compare	compares 2 files (any type)
copy	copies a file
cptree	copies a directory and its descendents to another directory
create	creates a file
crogen	generates a Cromix Operating System

day	executes a command on the day specified
dcheck	checks the internal structure of a directory
ddump	converts and copies data from one file or device to another.
default	sets the default root device and login name
delete	removes a file or directory from a file system
deltree	deletes a directory and its descendents
directory	changes or displays the current directory
dump	displays the contents of a file in hexadecimal
echo	sends its argument to the console
exit	exits from a Shell and/or logs the user off
find	finds files
fixsb	command file that restores the Superblock
free	displays the amount of unused space on a device
goto	transfers control within a command file
help	displays the online manual
icheck	checks the integrity of a file system
idump	displays the contents of an inode
if	conditionally executes a command within a command file
init	initializes a disk by erasing all data on it
input	reads a line from standard input and sends it to standard output.
ioprun	loads a program into an IOP
kill -1 1	consults the ttys file for changes
kill -2 1	kills all processes and shuts down the system

kill 0	kills all detached processes started from your terminal
kill	sends a kill signal to a process
l	lists information about a file
mail	handles mail between users
makdev	creates a device file
makdir	creates a directory
makfs	makes a file system
maklink	makes a link to a file
match	finds all occurrences of a string within a file
mode	displays or alters character device modes
mount	connects a file system (disk) to the current file system
mounthelp	mounts the second Cromix Operating System diskette
move	moves a file from one directory to another
msg	sends a message to another user
ncheck	displays file information
newdisk	copies the contents of the root device to a blank disk
newuser	displays information of interest to a new user
passwd	changes a user password, adds, or deletes a user
patch	patches files
path	shows the path to a specified command
pri	changes the priority of a process
priv	changes user status to that of a privileged user
prompt	changes the prompt to a specified character

pstat	displays the status of a process
query	locates Shell commands and utility programs
rename	changes the name of a directory or a file
repeat	repeats a command a specified number of times
restore	restores data saved by the Backup utility
rewind	restores arguments within a command file to their original positions
root	displays the device containing the root directory
runqd	reconfigures the system to use an IOP/Quadart console channel
runtu	reconfigures the system to use a 16FDC console channel
screen	calls the Screen Editor for editing files
shell	creates a Shell process
shift	uses the next command line argument from within a command file
shutdown	shuts down the Cromix Operating System
sim	allows CDOS programs to run under the Cromix Operating System
sleep	puts a process to sleep for a specified number of seconds
sort	sorts or merges files
spool	queues files and sends them to a printer
startup	contains commands executed every time the system is started up
tee	pipes output to a file as well as to the standard output
testinp	tests the contents of a file for a particular string or strings
time	displays or alters the time and date

- **type** sends the contents of a text file or standard input to the standard output
- **unmount** disconnects a file system (disk) from the current file system
- **update** updates a disk with a newer version of the Cromix Operating System
- **usage** displays directory size information
- **version** displays the version number of the Cromix Operating System or utility
- **wait** waits until all detached processes have finished
- **wboot** initializes the boot track of a floppy disk
- **who** lists the users presently logged in
- **>** redirects the standard output to a file
- **>>** appends the standard output
- **<** redirects the standard input from a file
- **>*** redirects the standard output and standard error to a file
- **><** sequentially pipes the standard output only
- **><*** sequentially pipes the standard output and standard error
- **>>*** appends the standard output and standard error to a file
- **|** pipes the standard output only
- **|*** pipes both the standard output and standard error

utility:	**ACCESS**	
purpose:	This program changes the access privileges associated with a file.	
user access:	all users	files owned by the user
	privileged user	any file
summary:	access [+rewa].[+rewa].[+rewa] file-list	
arguments:	access privilege specifier string	
	one or more pathnames	
options:	none	

Description

The Access utility allows a user to change file access privileges.

The access privilege specifier string (first argument) contains three clusters of access flags separated by periods. The first cluster indicates owner permitted access, the second indicates group access, and the third indicates public access. Each cluster is composed of zero or more of the following flags, given in any order:

+ add the specified privileges
r read access
e execute access
w write access
a append access

Refer to the discussion of file protection in Chapter 3 for additional information.

Notes

The Access utility allows the user to change file access privileges in several different ways. The first of these is to reenter each access privilege for each population segment, making the desired changes. For example:

```
%access rewa.rw.a xyz
%l -l xyz
312     1 rewa r-w- ---a joe          Mar-09 18:25  xyz
```

The second method for specifying access privileges involves the use of the plus sign (+) in one or more of the access population clusters. When used in this manner, the plus sign means that the attributes for the specified population segment remain the same. The plus sign may also be followed by access privileges to be added for the given population segment.

```
% l -l abc
517     1 rewa re--  re--  joe        Mar-09 18:26  abc
% access +.+.+a  abc
% l -l abc
517     1 rewa.re.re-a     joe        Mar-09 18:30  abc
```

utility:	**BACKUP**
purpose:	This utility copies a directory and all subdirectories and files to a block device.
user access:	all users
summary:	backup [-tv] source-dir dest-dev [file-list]
arguments:	source directory destination device one or more filenames (optional)
options:	-t time -v verbose

Description

The Backup utility copies the source directory along with all descendant directories and files to the destination device.

Disks to be used with the Backup utility should first be initialized for use with the Cromix Operating System. It is not necessary to make a file system (Makfs) on the destination device. **Note that data existing on the disk in the destination device is destroyed.**

If the destination device is a floppy disk and all the data does not fit on one floppy disk, the Backup program prompts for additional disks.

Only files whose names match at least one of the names in the file list are backed up. Ambiguous filenames enclosed in quotation marks may be included in the file list.

Backup does not modify dump times.

Options

The **-t** option backs up a file **only** if the source file was modified since the last back up.

The **-v** option causes the names of all files to be displayed as they are backed up.

Notes

The data that has been backed up may be restored only by the Restore utility. A disk written by the Backup utility may be accessed **only** by the Restore utility.

Modifying the source directory while Backup is in progress can result in a **phase error.**

The disk in the destination device must **not** be mounted. (Do not use the Mount utility to mount the disk).

Example:

```
# backup -v fda

  filea
  fileb
  filec
  filex
```

utility:	**BLINK**
purpose:	This program links relocatable files.
user access:	all users
summary:	blink [-dinpqrxz] [-b outname] filename [-s libname] . . .
arguments:	one or more filenames optional library name following each filename
options:	-b output file name -d data section address -i IOP starting address -n no map -p program address -q do not display map -r relocatable binary -s search library -x bitmapped -z size (use with -r)

Description

The Blink utility is a two pass virtual linker. One or more input files can be specified. An executable binary file is generated. Blink can be used to generate relocatable binary files which can share a bank of memory with other programs.

Options

The **-b** option may be used to specify the output filename. If used, the **-b** option must be followed by a space and the name of the binary file to be created. If this option is not used, the output file adopts the name of the first relocatable file specified on the command line. The output file has the filename extension **.bin.** This option may be used to force the output file to have a filename extension of **.com.** These are programs compatible with the CDOS operating system only if they were written using CDOS system calls. The format for linking these files is:

```
% blink -b filename.com modulenames
```

The **-d** option is followed by a space and the hex value of the data section starting address.

The **-i** option is followed by a space and the hex value of the starting address for an IOP program. It allows relocation of the program above the memory area occupied by the IOP Monitor. The IOP Monitor occupies memory between addresses 0000 and 0800 hex in ROM, and between 7F00 and 7FFF hex in RAM. This option creates an automatic header for the program to be run in the IOP using the Ioprun utility program.

The **-n** option prevents creation of a link map. Otherwise, the link map is created and written to a file with the filename extension **.map**.

The **-p** option must be followed by a space and the hex value of the program starting address. If no starting address is specified, the program starts at 100 hex. A relocatable binary program is placed wherever there is space in a memory bank.

The **-q** option inhibits display of the link map on the terminal. Otherwise, the link map is displayed on the terminal.

The **-r** option causes generation of the output file in relocatable binary format. Programs in this format can be executed with another process in a single bank of memory. The **-r** option is used with the **-z** option discussed below.

The **-s** option precedes the filename of the library to be searched. The option applies only to the file immediately following it, and must be specified for each file to which it applies. Blink searches the **.rel** file for necessary functions. If no library is specified using the **-s** option, and there is no library in the current directory, the program looks into **/usr/lib**, which is the default system library directory.

The **-x** option makes the output file a bitmapped self-relocating file. This option generates a self-relocating file which, when loaded into a user bank, loads in highest available memory and sets high memory to the byte just below itself. This option is used in linking the Cromemco Debug program.

The **-z** option allocates a specific size for the program segment. This switch is used only with the **-r** option, and only when free space (more than Blink normally allocates) is desired in the program area.

Notes

Blink manages memory so as to link programs up to the total amount of memory available. The memory area used by the linker during execution does not impose a restriction on the size of the program being linked. Thus, Cromix programs up to 64K, minus 1K of memory occupied by the Cromix Operating System in each user bank, can be linked by Blink.

CDOS programs running under the Cromix Operating System are limited to approximately 4K less memory than the 63K available to Cromix programs. This is because Sim, the CDOS simulator, must also be loaded.

COBOL programs using segmentation cannot be linked with Blink.

utility:	**BOOT**
purpose:	This utility loads an operating system into memory.
user access:	privileged user
summary:	boot [filename]
arguments:	filename (optional)
options:	none

Description

The Boot utility loads an operating system into memory.

If no argument is given, the file **/cromix.sys** is loaded, and execution begins. In this manner, the Boot utility can be used to warm boot the Cromix Operating System.

Example:

```
# boot

Floppy = 1, Hard disk = 2
Enter major root device number: 2

hd0 = 0, hd1 =1, hd2 =2
enter minor root device number: 0
```

Here, the Boot utility is executed and the Cromix Operating System reloaded. The root device is specified as hard disk (2) number 0 (0).

If Boot is followed by a filename, the file is assumed to have a **.sys** extension. If the user needs to boot CDOS from the Cromix Operating System, the file **cdos.com** can be copied to the root directory using the Cdoscopy utility. The file must be renamed **cdos.sys**. The user then types **boot /cdos** to load CDOS and begin execution under CDOS.

Notes

Because this program loads an operating system, it interrupts any active processes. Be sure that no one else is executing a program and that there are no detached processes running on the system before executing the Boot utility. Otherwise, data may be lost.

One quick method to determine if there are users on the system is to execute the program status (Pstat) command:

```
# ps -a

  PID State Command
    1   S    -
  112   R    Shell
  105   R    screen letter
   18   S    login 1 19200 tty6
   94   S    shell
   16   S    shell
   15   S    shell
   14   S    shell
   89   S    login 1 9600 tty1
```

Here the Pstat command is executed with the **all** option by entering **ps -a** after the Cromix prompt. The display shows one user running the Screen Editor program to edit a file named **letter.** If the Boot program was executed at this point, the user would lose all editing changes made during this session.

As long as all lines of the Pstat display show a command of **shell** or **login,** no processes are running and it is safe to load an operating system.

The Boot utility may be executed only by a privileged user.

utility:	**CDOSCOPY**
purpose:	This utility copies files to and from CDOS disks.
user access:	all users
summary:	cdoscopy [-belvw] devname file-list
arguments:	Cromix device name name(s) of the file(s) to be copied
options:	-b binary file -e erase file -l list CDOS directory -v verbose -w write CDOS file

Description

The Cdoscopy utility copies files from a Cromemco Disk Operating System (CDOS) format disk to a Cromemco Cromix Operating System format disk and vice versa. For example:

```
% cdoscopy fdb letter
% cdoscopy -w sfda notes
```

The first of these command lines copies a CDOS file named **letter** (located on a large floppy disk in drive B) into the user's current directory. The second command line copies the Cromix file named **notes** from the user's current directory to a small floppy disk in drive A. In the first case, the file is converted from a CDOS format to a Cromix format. A Cromix format to CDOS format conversion takes place in the second example.

The Cromix Operating System **cannot** read CDOS disks. Programs to be executed and data to be read under the Cromix Operating System **must** be transferred from CDOS formatted disks to Cromix formatted disks before execution begins.

Where a file pathname is specified, CDOS considers the lowest level filename. This is the portion of the pathname to the right of the rightmost slash. For instance, the following command line puts the file named **memo** onto the CDOS format disk in drive B.

```
% cdoscopy -w fdb /usr/mary/memo
```

Options

The **-b** option copies binary files. When this option is used, the 1Ah (end of file mark) is not stripped from the end of the file.

The **-e** option erases the specified file(s) from the CDOS disk.

The **-l** option displays the contents of the CDOS directory.

The **-v** option displays files while they are copied to and from CDOS disks.

The **-w** option causes the file to be written to the CDOS disk.

Notes

When an ambiguous CDOS file reference is used, it must be enclosed in quotation marks.

The file **/usr/lock** must be present to execute the Cdoscopy program.

Examples:

```
% cdoscopy -v fda "*.z80"
% cdoscopy -vw hdl **
% cdoscopy -l fdb
```

These examples assume that the disks in drive A (fda) and B (fdb) and the hard disk (drive F or hdl) are CDOS disks. The first example copies all CDOS files on drive A having the filename extension **z80** into the current directory. The ambiguous CDOS file reference was placed inside quotation marks.

The second example writes all files in the current directory <u>to</u> the CDOS hard disk designated as F (Cromix Operating System designation **hd1**). No quotation marks were used for the Cromix Operating System ambiguous file reference.

The final example displays the directory of the CDOS disk in drive B (Cromix file designation **fdb**).

Refer to Appendix D for a list of device names.

utility:	**CHECK**
purpose:	This program runs the Dcheck and Icheck utilities.
user access:	privileged user
summary:	check [-s] [devname]
arguments:	optional device name
options:	-s

Description

The Check command runs the programs Dcheck and Icheck on a file system. Check should be run after rebooting the system or any time the integrity of the file system is in doubt. The Startup command file program executed after every boot up indicates when the Check program needs to be run. See the Startup command file description in this chapter for more information on Check.

Options

The -s option is the salvage option used with Dcheck and Icheck to repair most file system problems. See the description of the Dcheck and Icheck utilities in this chapter for more information. The system is rebooted after running Check with the salvage option.

utility: **CHOWNER**
purpose: This program changes the owner or group of a file.

user access: privileged user

summary: chowner [-gv] ownername file-list

arguments: name or number of the user to whom ownership is to be transferred

or

name or number of the group to which ownership is to be transferred

and

one or more filenames

options:
-g change group
-v verbose

Description

The Chowner utility changes the owner or group associated with any type of file. If the file **abc** is in the current directory and is owned by **mark**, the L utility might display it as:

```
# l -l abc
27     1 rewa re-- re-- mark         Mar-11 19:59  abc
```

Using the Chowner utility, ownership can be transferred to **cindy**:

```
# chowner cindy abc
# l -l abc
27     1 rewa re-- re-- cindy        Mar-11 19:59  abc
```

Options

The **-g** option allows the Chowner utility to change the group associated with the file. This option is used in the manner previously described, substituting the group name for the owner name.

The **-v** option displays the name of each file as its ownership is changed.

Notes

When the ownership of a file is changed, the group with which the file is associated changes to that of the new owner.

utility:	**CMPASC**
purpose:	This program compares two ASCII (text) files.
user access:	all users
summary:	cmpasc file1 file2
arguments:	2 filenames
options:	none

Description

The Cmpasc utility compares two ASCII (text) files and reports differences in content. Differences are shown by displaying the text of the first file, followed by the corresponding line in the second file which differs from the first.

Notes

The Cmpasc utility adjusts for internal differences in the two files (insertions or deletions).

Example:

```
% cmpasc fileone filetwo
-----> fileone
This file is sample file one.

-----> filetwo
This file is sample file two.
```

utility:	**COMPARE**
purpose:	This program compares two files.
user access:	all users
summary:	compare file1 file2
arguments:	2 filenames
options:	-t terse

Description

The Compare program compares two files and reports differences in length and content.

Compare lists differences between the files on a byte-by-byte basis. It displays an address in hexadecimal, then the byte in the first file at that address, followed by the corresponding byte in the second file. Compare does not adjust for offset, should one file lack one or more bytes in the middle (e.g., if part of a file was deleted). Use the Cmpasc utility to compare ASCII files.

Options

The **-t** option suppresses the list of differences. When this option is used, only a message is displayed to indicate whether the files are the same or different.

utility:	**COPY**
purpose:	This utility copies a file.
user access:	all users
summary:	copy [-dftv] source-file destination-file [-dftv] file-list dirname
arguments:	two single file pathnames **or** one or more file pathnames **and** a directory pathname
options:	-d directory & device files ok -f force -t time -v verbose

Description

The Copy utility copies one or more files. It does not alter the source file(s).

In its simplest format, copy duplicates file **abc** as file **xyz**, with both files residing in the current directory:

```
% copy abc xyz
```

To copy to or from a directory other than the current directory is more complex:

```
% copy abc /usr/fred/xyz
```

Here the pathname of the destination file is specified. The file **abc** exists in the current directory. It is being copied to the directory **/usr/fred** and its name is to be **xyz** in that directory.

In the command:

```
% copy abc /usr/fred
```

the pathname of the destination directory is specified. The file **abc** exists in the current directory and is being copied to the directory **/usr/fred** without having its name changed.

The following form of the command can be used to create an archive of all C language programs in a given directory:

```
% copy *.c /usr/archives
```

This Copy command copies all files in the current directory with filenames ending in **.c** to the directory **archives.** The files maintain their original names.

Options

The **-d** option allows directory and device files to be copied. If this option is not used, directory and device files are not copied. For example, a command such as:

```
copy -d /dev/tty2 data
```

can be used to transfer all characters typed at terminal 2 into the file named data until a terminating character is received. The terminating character for console devices is CNTRL-Z.

The -f option makes the copied file overwrite an existing file with the same pathname. If this option is not specified and another file exists with the destination pathname, an error is reported.

The **-t** option causes a file to be copied **only** if:

1. The file does not exist in the destination directory; or

2. The source file has been modified more recently than the destination file. This comparison is performed on a file-by-file basis.

The -v option displays the name of each file as it is copied.

utility:	**CPTREE**
purpose:	This program copies a tree.
user access:	all users
summary:	cptree [-ftv] source destination [file-list]
arguments:	source directory destination directory optional file list
options:	-f force -t time -v verbose

Description

The Cptree utility copies the source directory, and all its descendant directories and files to the destination directory. Existing links within the source directory are preserved.

If a file list is specified, only files whose names match at least one of the names in the list are copied. Ambiguous filenames enclosed in quotation marks may be included in the file list.

Options

The **-f** option causes the copied files to overwrite any file with the same pathname. If this option is not invoked and another file exists with the destination pathname, an error is reported.

The **-t** option causes a file to be copied **only** if:

1. the file does not exist in the destination directory, or

2. the source file has been modified more recently than the destination file. This comparison is performed on a file-by-file basis.

The **-v** option causes display of the name of each file as it is copied.

Shell command:	**CREATE** or **CRE**
purpose:	This command creates a file.
user access:	all users
summary:	cre file-list
arguments:	one or more pathnames
options:	none

Description

The Create command is used to create one or more files.

The files are zero bytes in length and have default access privileges. They are owned by the user who created them and are in the domain of their creator's group.

If the specified pathname already exists, an error is reported.

Notes

This command makes a standard data file. Refer to the Makdir command or the Makdev utility program if you need to make a directory or device file.

utility:	**CROGEN**
purpose:	This program generates a Cromix Operating System.
user access:	privileged user
summary:	crogen -vb [pathname]
arguments:	optional pathname
options:	none

Description

The Crogen utility generates a new operating system. It allows the user to add and delete system drivers to provide the largest possible number of system buffers and Shells. The user may add user-defined character drivers to the operating system.

Crogen is a menu driven utility residing in the **/gen** directory. To use Crogen, select the **/gen** directory and begin execution of Crogen by giving the commands:

```
# d /gen
# crogen
```

Crogen displays the prompts shown below. To give the default response, enter RETURN.

CHARACTER DEVICE DRIVERS

1 - Console (Tuart) (Y = Yes, N = No) <Y> ?

Are the standard tty drivers (16FDC, 4FDC, and TU-ART) to be included in this system? Respond with **Y** (for yes) or **N** (for no).

2 - Console (Quadart) (Y = Yes, N = No) <Y> ?

Are the IOP/Quadart drivers to be included in this system? Respond with **Y** or **N**.

3 - System must be present

This is the main Cromix module and must be included. No user response is required for this driver.

4 - Timer (Y = Yes, N = No) <Y> ?

Is the operating system clock to be included in this system? This question should always be answered **Y** except when another clock is being added.

5 - Parallel printer (Y = Yes, N = No) <Y> ?

Is the dot matrix printer driver to be included in this system?

6 - Typewriter printer (Y = Yes, N = No) <Y> ?

Is the fully formed character printer driver to be included in this system?

7 - Serial printer (Tuart) (Y = Yes, N = No) <N> ?

Is the ON/XOFF serial printer driver which interfaces with TU-ART included in this system?

8 - IOP Memory (Y = Yes, N = No) <N>?

Is this system to have a driver that allows you to read IOP memory for debugging purposes? The default response here is No. Even though you have an IOP, you do not need to have an IOP memory driver.

9 - Serial printer (Quadart) (Y = Yes, N = No) [N] ?

Is the XON/XOFF serial printer driver which interfaces with Quadart included in this system?

10- SDI (Y = Yes, N = No) <No>?

Is this system to have a graphics interface?

11- Tape (Y = Yes, N = No) <No>?

Is this system to have a TDS tape driver?

BLOCK DEVICE DRIVERS

Disk drivers to be included in this system?

If the answer is **Y**, Crogen responds with the following two questions:

1 - Floppy disk (Y = Yes, N = No) [Y] ?

Are floppy disk drivers to be included in this system? Respond with **Y** or **N**.

2 - Hard disk (Y = Yes, N = No) [Y] ?

Are hard disk drivers to be included in this system? Respond with **Y** or **N**.

DEFAULT ROOT DEVICE

Should the system automatically select a root device number? Respond with **Y** or **N**. If you answer **Y**, Crogen responds with the following question:

Major device number (1 = Floppy, 2 = Hard disk) [2] ?

Is the root device a floppy disk or a hard disk? Respond with **1** or **2**.

If the major device is a floppy **(1)**, Crogen responds with the following question:

```
Minor device number   (0 = fda     4 = sfda)
                      (1 = fdb     5 = sfdb)
                      (2 = fdc     6 = sfdc)
                      (3 = fdd     7 = sfdd)  [0]?
```

If the major device is a hard disk **(2)**, Crogen responds with the following question:

Minor device number (0 = hd0, 1 = hd1, 2 = hd2) [0] ?

Which hard disk should be the root device? Respond with **0, 1,** or **2.**

Automatic login name [none] ?

Should this system automatically login when the system is booted? Respond with the login name if this function is desired or press RETURN if it is not desired.

Default access for created files [rewa.re.re]?

All files created under this Cromix system initially have these access privileges.

Crogen now responds with the following message:

Creating cromix.sys (or other filename if specified)

This indicates Crogen is creating the new operating system and writing it to the specified file. In this example, Crogen has written the operating system to the current directory (**/gen/cromix.sys**).

If the optional pathname for Crogen is **Crogen /cromix,** Crogen overwrites the operating system in the root directory. If the new operating system is not correctly configured, it will not be possible to cold boot the system. Therefore, it is suggested that the new operating system be created in the **/gen** directory, tested by booting that operating system (**boot /gen/cromix**), and then moved into the root directory.

utility:	**DAY**
purpose:	This program executes a command on the day specified.
user access:	all users
summary:	day [day-of-the-week command-line]
arguments:	day of the week command line
options:	none

Description

The Day utility executes a command on the day specified. Day checks the system clock for the specified day. This program is useful in applications that require certain tasks be done on certain days of the week.

Notes

When used without an argument, Day displays the name of the current day.

Example:

The following command line will remind you of a weekly Wednesday meeting.

%day wed echo "This is Wednesday, remember your meeting"

utility:	**DCHECK**
purpose:	This program verifies the integrity of a file system.
user access:	all users
summary:	dcheck [-s] [devname]
arguments:	optional device name
options:	-s salvage directory structure

Description

The Dcheck utility verifies the integrity of a file system's internal directory structure. If possible, Dcheck with the salvage option should be run on an unmounted file system. If the file system that needs to be fixed is the root, Dcheck should be run by itself, with no other users or tasks running concurrently. If another task is writing to the disk, the results of Dcheck may be incorrect.

If the -s option is used while another task or user is using the disk, the <u>directory on the disk may be irreparably damaged</u>.

MESSAGES RETURNED BY DCHECK

Cannot read super block
The super block cannot be read.

Out of memory
The disk contains too many inodes for Dcheck to check. Make a new disk with fewer inodes and use the Cptree utility program to transfer the contents of the disk to the new disk.

Cannot read inode xxxxx
A disk I/O error occurred while trying to read the specified inode.

Inode xxxxx, error reading directory
A disk I/O error occurred while trying to read a directory.

Inode xxxxx, cannot read inode
A disk I/O error occurred while trying to read the specified inode.

Inode xxxxx, directory with more than 1 parent
A directory is linked to more than 1 parent directory. Use the Ncheck utility program to locate the names of the files and delete all but one link. Then run Dcheck with the **-s** option.

Inode xxxxx, directory with wrong parent
This error indicates the inode is pointing to the wrong parent. Use the Dcheck utility with the **-s** option to correct this error.

Inode xxxxx, bad link count xxxxx, should be xxxxx
The number of names pointing to this inode from various directories is greater or less than expected. Use Dcheck with the **-s** option to correct this error.

Inode xxxxx, more than 255 links
There are more than 255 names for this inode. Use Ncheck to find all the names. Delete some names to bring the total number of names to 255 or less, then run Dcheck with the **-s** option.

Inode xxxxx, bad inode number in inode
Each inode contains its own inode number. This error means the inode specified has the wrong number. Use Dcheck with the **-s** option to correct this error.

Inode xxxxx, unallocated inode with xxx links
Although this inode is unallocated, names point to it. Use Ncheck to find these names, then delete them.

Inode xxxxx, allocated inode with 0 links
This inode is still allocated, though there are no names for it. Use Dcheck with the **-s** option to correct this error.

Inode xxxxx, bad directory entry count
This inode is a directory. The number of directory entries in the inode differs from the actual number of directories. Use Dcheck with the -s option to correct this error.

End of Dcheck (This is the last message)

The program has finished executing.

Options

The -s option fixes problems reported by Dcheck. The program corrects an incorrect inode number when:

1. The inode is allocated;

2. The inode link is nonzero; and/or

3. The inode is being pointed to (i.e., is in use).

The program does not correct an incorrect inode number if the inode is unallocated.

Notes

Immediately after running Dcheck with the -s option, run Icheck with the -s option. After both programs are run, the system must be rebooted. Refer to the Boot utility for additional information.

It is not necessary to reboot if the -s option is not used.

utility:	**DDUMP** (Direct dump)
purpose:	This program converts and copies a file from one device to another. It can handle direct physical I/O from devices such as the tape driver.
user access:	all users
summary:	ddump options

Option	Function
if=pathname	specify input file pathname
-i pathname	specified input file pathname
of=pathname	specify output file pathname
-o pathname	specified output file pathname
ibs=n	input block size
obs=n	output block size
cbs=n	conversion buffer size
cbufsz=n	conversion buffer size
iskip=n	skip the first n input blocks before starting to copy
oskip=n	skip the first n output blocks before starting to copy
icount=n	copy only n input blocks
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ucase	convert alphabetic characters to upper case
lcase	convert alphabetic characters to lower case
strip	strip trailing blanks in the conversion buffer
nostop	do not stop processing on an error (such as a file read error)

Several conversions, separated by commas, may be specified in one argument.

Description

Ddump converts and copies data from one file or device to another. Since the input and output block sizes can be specified, it is useful for gaining access to devices that store data in raw form.

Conversions are done in the conversion buffer. Each block read from the input file is transferred to the conversion buffer, one buffer at a time. The conversions specified are performed there before writing the result to the output file. For example, if the **strip** conversion option is specified, trailing spaces are stripped and a newline added before sending the result to the output file.

Example:

```
# ddump if=/dev/tpl of=filel conv=ascii,lcase,strip
```

This example causes input to be read from **/dev/tpl** and written to disk file **filel.** EBCDIC characters are converted to ASCII, uppercase to lower, and trailing blanks are not copied to **filel.** The end of the tape file is indicated by an EOF tape mark written when the tape was created.

Notes

The following is a list of default values for options.

input file	standard input
output file	standard output
conversion buffer	80 bytes
disk input buffer	512 bytes
disk output buffer	512 bytes
tape input buffer	8192 bytes
tape output buffer	8192 bytes

utility: **DEFAULT**
purpose: This program sets default parameters for automatic boot up and log in.

user access: privileged user

summary: default pathname majornum minornum [login-name]

arguments: pathname

major device number

minor device number

optional login name

options: none

Description

The Default utility allows the Cromix Operating System to use a default root device and login name when booting the operating system. If the login name requires a password, the user is prompted for it; otherwise the boot and login procedure execute automatically.

Pathname specifies the directory and filename of the **cromix.sys** file to be changed. The device number is that of the default root device (refer to Table 6-1).

Notes

This program is not in the **/bin** directory, but in the **/gen** directory.

If the major and minor device numbers are both zero (0), then no default device is established and a prompt for the root device is displayed when the system is booted.

Shell command:	**DELETE** or **DEL**
purpose:	This command deletes a file.
user access:	all users
summary:	del [-v] pathname(s)
arguments:	one or more pathnames
options:	-v verbose

Description

The Delete command removes a link to a file. If there is only one link to the file, the file is no longer accessible and the space it occupied is made available.

Options

The -v option displays the name of each file as it is deleted.

Notes

To remove all links to a file, making it inaccessible, use the L command with the -i option to find the inode number of the file in question. Use that inode number as an argument to Ncheck, and find the names of all files linked to the file.

A directory may be deleted by specifying a directory pathname.

In order to delete a directory, it must not:

1. Contain any files;

2. Be the current directory for any user; or

3. Be the root directory of a device.

Examples:

In the following example, the file named **schedule** is deleted from the current directory.

```
% l

      3,016    1 letter
        200    1 memo
      1,408    1 schedule

% del schedule
% l

      3,016    1 letter
        200    1 memo
```

If there is more than one link to a file and one of the links is deleted, the file is no longer accessible through that link. The file remains on disk and is accessible through the remaining links.

The following example concerns itself with part of the **/dev** directory. As the Cromix Operating System is shipped, the dummy file **prt** is linked to the dot matrix printer driver **lptl**. In the first listing that follows, the link is shown by the **2** preceding each filename. When the file **prt** is deleted, the file **lptl** remains intact and the number of links is reduced to one.

```
# l

5:5  C     2 lptl
5:5  C     2 prt
6:5  C     1 typl

# del prt
# l

5:5  C     1 lptl
6:5  C     1 typl
```

utility:	**DELTREE**
purpose:	This program deletes a tree, including all files and subtrees.
user access:	all users
summary:	deltree [-a] pathname
arguments:	pathname
options:	-a suppresses user verification

Description

The Deltree utility deletes all files and subtrees in the tree (directory) specified. Normally, Deltree prompts the user with the file or directory name and (y,n). If the user types **y**, the file or directory is deleted; otherwise it is not. If the **-a** option is used, Deltree asks once whether the user really wants to delete the entire tree, instead of prompting for verification of each file. If the user types **y**, all files and subtrees are deleted. If **n** is typed, Deltree returns to the Cromix prompt.

If Deltree is called from within the specified directory, the program will not allow the deletion of that directory. All the files must be deleted from a directory before the directory itself is deleted.

Options

After asking for verification, the **-a** option deletes all files and subtrees automatically.

Shell command:	**DIRECTORY** or **D**
purpose:	This command displays the name of or changes the current directory.
user access:	all users
summary:	d [dir name]
arguments:	optional directory pathname
options:	none

Description

When given without an argument, the Directory command displays the pathname of the current directory.

Given with a directory pathname, the Directory command makes the specified directory the current directory.

utility:	**DUMP**
purpose:	This program displays a file in hexadecimal and ASCII.
user access:	all users
summary:	dump [-b #] file-list [-e #] [-k #] [-s #] [-o #]
arguments:	one or more file pathnames
options:	-b first byte -e last byte -k first block -s width -o offset address

Description

The Dump program displays the file(s) specified by the pathname(s). Dump displays any type of file. The file is displayed in hexadecimal with an ASCII equivalent to one side. All numeric arguments to the Dump utility are assumed to be decimal numbers unless followed by an **h** (for hexadecimal).

Options

The **-b** option allows the user to specify the first byte of a file to be dumped.

The **-e** option allows the user to specify the last byte of a file to be dumped.

The **-k** option allows the user to specify the first block to be dumped.

The **-s** option allows the user to specify the swath width of the dump.

The **-o** option causes a specified offset to be added to all addresses displayed by Dump.

Example:

% dump -b 1000h -e 5000h filename

This command dumps the file **filename** starting with the 1000th (hex) byte and ending with the 5000th (hex) byte.

utility:	**ECHO**
purpose:	This program echoes its arguments to the terminal.
user access:	all users
summary:	echo text
arguments:	any text
options:	-e send to stderr -n do not print newline

Description

The Echo program echoes its arguments. Text may be enclosed within single or double quotation marks to insure correct interpretation by the Shell. Echo is a relocatable binary program.

Options

The **-e** option echoes arguments to the standard error channel.

The **-n** option suppresses the echo of a newline.

Shell command:	**EXIT** or **EX**
purpose:	This command exits from a Shell.
user access:	all users
summary:	ex
arguments:	none
options:	none

Description

The Exit command is used to exit from a Shell. If no higher level Shell is active, the Cromix Operating System logs the user off the system.

utility: **FIND**
purpose: This program locates files.

user access: all users

summary: find pathname [!] expression

arguments: pathname

[!]

expression(s)

options: File specifiers:
- -name
- -type x
- -links n
- -user name or number
- -group name or number
- -size n
- -blocks n
- -mtime n

Action Specifiers:
- -exec command-line
- -ok command-line
- -print

Logical Operators:
- -a
- -o

Description

The Find utility locates a file. The pathname is the pathname of the tree, directory, or file to be searched, and the expression is the string to be found and what is to be done with it.

Expressions are combinations of file criteria and operations. Refer to the following list.

Parentheses may be used to change the order of evaluation of the items in the Find expression. Used with parentheses, the expression must be enclosed within quotation marks so that the Shell passes them to the Find utility.

When one of the action specifiers is used to execute a program, the return code of that program can be evaluated and used within the expression.

The **!** operator may precede the expressions to negate the sense of the tests.

Options

File Specifiers

-name file-list
: The file specifying keyword **name** is followed by a list of one or more unique or ambiguous filenames. If an ambiguous filename is used, it must be enclosed within quotation marks. The Find utility finds all files that match the file list.

-type b block device
      c character device
      f file
      d directory

The file specifying keyword **type** is followed by either **b**, **c**, **f**, or **d**, as shown. The Find utility finds all files of that type.

-links n
: The file specifying keyword **links** is followed by a number, **n**. The Find utility finds all files with that number of links. If the number is preceded by a plus sign, all files with more than that many links are found; if a minus sign is used, all files with fewer than **n** links are found.

-user name
      number
: The file specifying keyword **user** is followed by a user name or number. The Find utility finds all files owned by the specified user.

-group name
       number
: The file specifying keyword **group** is followed by a group name or number. The Find utility finds all files owned by the specified group.

-size n — The file specifying keyword **size** is followed by a number, **n.** The Find utility finds all files of the specified size in bytes. If the number is preceded by a plus sign, all files with more than that number of bytes are found; if a minus sign is used, all files with fewer than **n** bytes are found.

-blocks n — The file specifying keyword **blocks** is followed by a number, **n.** The Find utility finds all files using that number of blocks (actual number of blocks occupied by the file). If the number is preceded by a plus sign, all files occupying more than the specified number of blocks are found; if a minus sign is used, all files with fewer than **n** blocks are found.

-mtime n — The file specifying keyword **mtime** is followed by a number, **n.** The Find utility finds all files modified **n** days ago. If the number **n** is preceded by a plus sign, all files modified **n** or more days ago are found; if a minus sign is used, all files modified fewer than **n** days ago are found.

Action Specifiers

-exec command-line

The action specifying keyword **exec** is followed by a command line. This may be any valid command line, that is, any line that can be entered in response to the Cromix prompt. This command line is then executed each time the Find utility finds a file meeting the find criteria. A pair of braces ({}) may be placed within the command line. They will be replaced by the name of the file found.

-ok command-line

The action specifying keyword **ok** is used in the same manner as **exec.** When **ok** is used, the Find utility prompts the user prior to executing each command line. The user may respond with a **y** to execute the command line, or **n** to prevent its execution.

-print — The action specifying keyword **print** is used to display the pathnames of files found.

Logical Operators

-a — The **-a** operator is used to logically **AND** two items in the Find expression.

-o — The **-o** operator is used to logically **OR** two items in the Find expression.

Notes

The expression used with the Find command is evaluated from left to right. Items to be found and actions to be performed may be combined logically by use of the **-a** and/or **-o** logical operators. Either operator combines the sum of the expression to its left with the subsequent item in the expression. For example:

```
find / -name ted -a -print

find / -name ted -o -name mary -a -print
```

The first example finds all files with the filename **ted** and prints the pathnames of these files. If the print instruction is left out of this command line, all of the correct files are found and no action is taken: their names are not displayed. The second example demonstrates the use of the logical OR. All files with the filename **ted** OR **mary** are found and their pathnames printed.

Examples:

The following example finds all subdirectories of the current directory, then executes an **l** command with the **-d** and **-e** options.

```
% find . -type d -a -exec l -de {}
```

The next example finds all entries with a .c extension, then lists the entry with the -l option.

```
# find / -name "*.c" -a -exec l -l {}
```

utility:	**FIXSB**
purpose:	This command file restores the Superblock.
user access:	privileged users
summary:	fixsb
arguments:	none
options:	none

Description

The Fixsb utility file restores the Superblock, should it be destroyed accidentally. This command file has the same function as the Makfs utility used with the -r option, but without the possible risks associated with running an older version of the Makfs utility.

After restoring the Superblock, the Fixsb command automatically runs Icheck, to check inodes in the file.

Notes

Fixsb is only to be used on disks whose file systems were created with the default number of inodes. Refer to the Makfs utility for additional information.

utility:	**FREE**
purpose:	This program displays the amount of unused space remaining on a device.
user access:	all users
summary:	free [devname1 ... devnameN]
arguments:	optional list of device names
options:	none

Description

The Free program displays the amount of unused space remaining on a specified device. If no device is specified, the free space is displayed for all mounted devices.

Example:

The following is a sample output of the Free utility. It shows the available free space in blocks, kilobytes, and bytes.

```
/dev/root    7,513 blocks    3,756K    3,846,656 bytes
```

Shell command:	**GOTO** or **GO**
purpose:	This command causes transfer of control within a command file.
user access:	all users
summary:	go label
arguments:	line label
options:	none

Description

The Goto command transfers control within a command file. Control is transferred to the line specified by **label.** This command is used to execute the same commands within a command file repeatedly. When used in conjunction with the If and Shift commands, the Goto command becomes part of a conditional loop with varying parameters.

A **line label** is any line within a command file that begins with a percent sign (%). If a percent sign appears as a character other than the first character on a line, the balance of the line is a **comment** and thus ignored by the Cromix Shell.

The Goto command given with a nonexistent line label causes termination of command file execution.

Example:

```
%sample_label
x
y
z % this is a comment
goto sample_label
```

This sample command file causes repeated execution of the commands **x**, **y**, and **z**. The first line of the command file is a line label, as indicated by the leading percent sign.

Notice that the percent sign indicates a comment on the fourth line of the file. The fifth (last) line of the file transfers control to the specified label (sample_label).

utility:	**H** or **HELP**
purpose:	This program displays pages on Shell commands and utility programs from the Cromix manual.
user access:	all users
summary:	help [command-name] [utility-name]
arguments:	optional command or utility name

functions:

b	beginning
h	help
q	quit
r	return to menu
u	up
RETURN	next line
space	next page

Description

The Help utility program provides a convenient online manual for user reference. The Cromix Instruction Manual entries for Shell commands and utility programs are the only accessible entries. Information regarding system calls and other aspects of the system must be obtained from the Cromix Instruction Manual.

Help can be called alone or with an optional program name. If unsure about the name of a utility or command, enter **help** and press RETURN. The Help program lists the available topics and asks you to select the desired topic.

Entering **help** with a program name lists the manual entry on that topic, omitting the list of available topics. The list of utilities and Shell commands may then be displayed by pressing the **r** key.

Help displays the Cromix manual entry one page at a time. The percentage of the file yet to be viewed is displayed at the bottom of the screen.

Several functions aid you in viewing the manual entry. Pressing the space bar displays the next page of the manual. Pressing RETURN displays the next line. Pressing **u** displays the previous page. The **u** key and the space bar can be used to move the user back and forth through the text. The **b** key causes a jump to the beginning of the manual. The **h** key displays a list of available functions for the Help program (Up, Beginning, Return, Quit, and Help). To exit from the Help program, press **q**.

Modifying the Online Manual

The database for the online manual is located in the **/usr/help** directory. Each topic is contained in a file with the name of the help topic and the filename extension **.hlp.** Additional topics can be entered in the **/usr/help** directory. The files must have the **.hlp** filename extension so that the Help program can gain access to them. The message written on the terminal above the listing of Help topics is found in the file named **/usr/help/help.msg** and can also be modified.

The file **/usr/help/msg.msg** contains the messages printed on the bottom line of the screen when the **help** program sends a file to the console. The **msg.msg** file is linked to the file **msg2.msg**, which contains messages taking advantage of the attributes of the Cromemco 3102 terminal. If your system uses the Cromemco 3100 or 3101 terminal, the file **msg1.msg** should be linked to the file **msg.msg** by entering the following command:

```
# maklink -f /usr/help/msg1.msg /usr/help/msg.msg
```

utility:	**ICHECK**
purpose:	This program verifies the integrity of a file system.
user access:	all users
summary:	icheck [-s] [-b blk# ...] [devname ...]
arguments:	optional list of device names
options:	-s salvage -b blocks

Description

The Icheck utility verifies the integrity of the file system's inode structure. After a power failure or after the computer has been reset, run Icheck on all mounted devices.

If no device names are specified, Icheck checks the integrity of all mounted devices. The list of mounted devices is obtained from the file **/etc/mtab**.

If no options are specified, Icheck produces a report on the file system, but does not alter it. A sample report and explanation follow.

> **If the -s option is used while another task or user is using the disk, the directory on the disk may be irreparably damaged.**

```
% icheck
Device: /dev/hd0

Blocks missing:                          0
Bad free blocks:                         0
Duplicate blocks in free list:           0
Bad blocks:                              0
Duplicate blocks:                        0

Device files:                           16
Ordinary files:                        269
Directories:                            44
Blocks used in files:               13,546
Indirect blocks:                       172
Free blocks:                         6,212
Free inodes:                         3,871
```

Blocks missing

All disks (also referred to as block devices) are divided into allocation units called **blocks.** A block is 512 bytes. Every block should appear either in a file or in the **free list.** Blocks appearing in files include those permanently assigned as either system or inode blocks. The free list is a list of all blocks available for use.

A block is **missing** if it appears neither in a file nor in the free list. Missing blocks do not compromise the integrity of the file system and the problem does not need to be corrected immediately. If a block is missing, it is simply not available for use.

The problem may be corrected by executing Icheck with the **-s** option.

Bad free blocks

This message pertains to blocks located in the free list. The term **bad** indicates that the block number is out of range. A block number can be out of range if it is:

1. Past the end of the disk;

2. In the system area of the disk; or

3. In the inode area of the disk.

Bad free blocks **do** compromise the integrity of the file system and the problem should be corrected immediately by executing Icheck with the **-s** option. No files are affected.

Duplicate Blocks in Free List

This message means the same block number appears twice in the free list.

Duplicate blocks in the free list **do** compromise the integrity of the file system and the problem should be corrected immediately by executing Icheck with the **-s** option. No files are affected.

Bad Blocks

This is similar to **Bad free blocks** except that the Bad blocks appear in files.

Bad blocks **do** compromise the integrity of the file system and the problem should be corrected immediately.

Icheck reports the inode number of the bad blocks. The Ncheck utility is then used to determine the names of the files containing bad blocks. These files must be deleted. The file may be copied to another file before it is deleted; the new file should be carefully checked because it will probably not be correct.

Duplicate Blocks

This is similar to **Duplicate blocks in free list** except that the Duplicate blocks appear in files.

Duplicate blocks **do** compromise the integrity of the file system and the problem should be corrected immediately.

Icheck reports the inode number of the duplicate blocks. The Ncheck utility is then used to determine the names of the files containing duplicate blocks. At least one of these files must be deleted. The Icheck utility should then be run with the **-s** option.

The file may be copied to another file before it is deleted and should be carefully checked because it will probably not be correct.

MESSAGES RETURNED BY ICHECK

Cannot read super block
The super block cannot be read.

Out of memory
The disk contains too many inodes for Icheck to check. Make a new disk with fewer inodes and use the Cptree utility program to transfer the contents of the disk to the new disk.

Cannot read inode xxxxx
A disk I/O error occurred while trying to read the specified inode.

Not a block device: "device name"
The device specified is not a block device.

Inode xxxxxx, ---- Bad usage count ----
This inode has an incorrect usage count. The usage count is used by the Usage utility program to calculate the amount of disk space used. This error can be corrected by running Icheck with the **-s** option.

Inode xxxxxx, ---- Cannot write to inode ----
This error message occurs when the Icheck utility is attempting to correct an inode and an error occurs.

Block xxxxxx, inode xxxxxx, ---- block used in file ----
This is not an error message. This message is displayed when the **-b** option is used, indicating the number of the inode in which the specified block is used.

Block xxxxxx, inode xxxxxx, ---- bad block number ----
Refer to the previous discussion of **Bad blocks.**

Block xxxxxx, inode xxxxxx, ---- duplicate block number
Refer to the previous discussion of **Duplicate blocks.**

Block xxxxxx, ---- block missing ----
This message is printed when the **-b** option is used to find the status of a certain block and the block is missing. Refer to the previous discussion of **Blocks missing.**

Block xxxxxx, ---- block in free list ----
This message is printed when the **-b** option is used to find the status of a certain block and the block is in the free list.

Block xxxxxx, ---- bad free block ----
Refer to the previous discussion of **Bad free blocks.**

Cannot write free list block xxxxxx
When running Icheck with the **-s** option, the free list is recreated. This error message is printed when there is an error in writing the free list.

Cannot read block xxxxxx
This message is printed when a block cannot be read.

Options

The **-s** option salvages and recreates the free list.

The **-b** option displays information about blocks.

Notes

When using the salvage option, Icheck must be used in conjunction with the Dcheck utility. Icheck is run after Dcheck. Both utilities should be run using the **-s** option. After both programs are run, the system must be rebooted. It is not necessary to reboot if the **-s** option is not used. Refer to the Boot utility for additional information.

Do not execute the Icheck utility when other processes are being executed. This includes detached processes as well as other user processes.

utility: **IDUMP**
purpose: This program displays the contents of an inode.

user access: all users

summary: idump blockdev inode-list

arguments: block device name

list of one or more inode numbers

options: none

Description

The Idump utility displays the contents of the specified inode(s).

Shell command: **IF**

purpose: This command is used to conditionally execute another command.

user access: all users

summary:

```
if -err command
   -rewa filename command
   string-1 = string-2 command
   string-1 != string-2 command
```

arguments: error condition specifier

or

access method and a filename

or

two strings separated by the equal (=) or not equal (!=) relational operator

and

a command line

options: none

Description

The If command is used to place a condition on the execution of another command. It is frequently used in conjunction with the Goto command and is terminated by a semicolon (;). Referring to the summary above, the If command has three basic forms.

The first form executes a command if the previous command returned an error.

In its second form, the If command causes **commands** to be executed if a particular access method applies to the file specified.

The third and fourth forms of the If command cause **command** to be executed when the specified relational condition is true or false. Neither of these forms of the If command requires that the strings be enclosed in quotation marks. However, both forms **do** require a space on either side of the relational operator (= or !=).

utility: **INIT**
purpose: This program initializes a disk.

user access: privileged user

summary: init

arguments: none

options: none

Description

The Init program is used to initialize disks.

Below is a sample script of a typical Init session to format a small (5-inch) Cromix floppy disk. The messages and questions are displayed by the Init program; the user's responses are descriptions of each part of the execution.

```
Initialize Disks version xx.yy

Press:   RETURN to supply default answers
         ESC to abort formatting
         CTRL-C to abort program
Warning: INIT can destroy all disk data

Disk to initialize (devname)?  sfdd

Testing:
     Index pulses being received correctly
     Rotational speed:  300 RPM

Formatting
     Disk type (C=CDOS, X=CROMIX)?  <X> RETURN
     Single or double sided (S/D)?  <D> RETURN
     Single or double density (S/D)> <D> RETURN

     First cylinder (0-27H)?  <OH> RETURN
     Last cylinder (0-27H)?  <27H> RETURN
     Surfaces (0-1,All)?  <All> RETURN
```

```
Cylinder, Surface:  restore
                    00H, 0
                    00H, 1
                    01H, 0
                    01H, 1
                     :   :
                     :   :
```

The Init program first asks for the device name of the drive containing the disk to be formatted. Legal responses are device names of the disk drives connected to your system, such as: fda, fdb, ..., sfda, sfdb, ..., hd0, hd1, ..., and so on. These may also be complete pathnames, such as: **/dev/fda, /dev/fdb,** and so on. **Be sure** to specify the device name correctly, as the Init program destroys all data on a disk.

Init briefly tests the specified drive to check if it is operating correctly for disk formatting. Since drive speed is particularly important to correct formatting, it is reported by the program. When formatting hard disks, the program also verifies that the controller board is working properly.

The next prompt asks about the type of disk to be formatted, that is, will it be used with the CDOS or Cromix Operating System? Either type may be formatted under either operating system.

If the disk to be formatted is a floppy, two more questions are asked about the combination of sides and density to be used in initialization.

Next, the Init program asks for the first and last cylinder numbers to be formatted. When an entire disk is being formatted, supply the default responses by pressing RETURN each time. On occasion, it may be necessary to format a portion of a disk (e.g., one track that seems to have frequent errors).

Next is a question about the surfaces to be initialized. The default response, that all surfaces are to be initialized, is made by pressing the RETURN key. However, you may format only one surface by typing its number, one of the values shown in the prompt.

Having been supplied with answers to all these questions, the Init program proceeds to format the disk. It displays its progress by giving cylinder and surface numbers. The disk is always formatted from the outermost cylinders inward, for proper head positioning.

Alternate Tracks

Since hard disks are designed for long term use and reliability, there is a provision for declaring alternate tracks, that is, good tracks to be used in place of tracks which develop hard errors. Once declared, the locations of these tracks are stored in a special area of track 0 (cylinder 0, surface 0) called the **alternate track table.** When a hard disk is formatted with Init, the program displays a list of previously declared alternate tracks and gives the user the chance to change these or declare new ones.

The display of alternate tracks usually prints **cyl xx, surf y** to indicate the cylinder and surface numbers of the track containing hard errors, or **unassigned** to show that the alternate track has not yet been used. The message **illegal entry** is a warning that, as far as the Init program is able to determine, an illegal value has been stored in the alternate track table. Whenever possible, this should be changed to a legal declaration.

Following the display of the alternate tracks originally defined, Init asks two questions: whether the user wishes to redeclare any of these already existing alternate tracks, and whether the user wishes to declare any new alternate tracks. It's usually best to keep the factory settings of the drive's alternate tracks unless you have a specific reason for changing them.

The Init program **cannot** and **should not** be aborted during the process of declaring alternate tracks for a hard disk. Certain information about the drive is in volatile memory during this time so aborting the program causes this information to be lost.

Finally, if you do choose to declare alternate tracks, the Init program prompts you with the alternate track number and requests the cylinder and surface numbers of the track containing hard errors. Alternately, you may press RETURN at this point and the alternate track declaration will remain the same. Or you may type **U** for unassigned, and a previously declared alternate track is freed up. Init can format both 8" and 5-1/4" hard disks.

Once a track is known to have hard errors and an alternate track is declared for it, Init makes no attempt to salvage the data stored on the bad track. It is best to recover as much of this data as possible before declaring the alternate track. The Hdtest program, supplied as a part of the Cromemco Diagnostics Software package (CDS), has special provisions for

recovering this data. The procedure using Hdtest to recover data is quite complex and best left to your authorized Cromemco dealer.

INIT ERROR MESSAGES

Incompatible with operating system
Use single-user or simulator CDOS xx.yy or higher

A version of the Init program is being used with an earlier, noncompatible version of CDOS or the CDOS simulator program running under the Cromix Operating System. Use the version of CDOS specified in the error message.

Initialization inhibited in this machine

Switch 4 of the 16FDC or 4FDC disk controller has been turned on. This switch prevents disk initialization in the computer system.

4FDC not capable of double density operation

The user has attempted to initialize a double density floppy disk in a system containing hardware for only single density operation. The system contains a 4FDC disk controller board; it requires a 16FDC.

Illegal device

The device name given in response to the **Disk to initialize?** question is not in the Cromix **/dev** directory.

Illegal Value

The number supplied in response to a prompt is illegal. This usually means the number is out of range.

Second number must equal or exceed first

This error appears when the response to the **Last cylinder?** question is not greater than or equal to the response to the **First cylinder?** question. The Init program always formats disks from the outermost cylinders inward to provide consistent head positioning.

Drive x is write-protected
Diskette in Drive x is write-protected

The drive or floppy disk specified has been write protected and cannot be initialized until the write protection has been defeated.

Drive x not ready

The drive specified is not ready, which usually means it cannot be selected by the software. This occurs when a floppy disk has not been properly inserted in its drive or the door has not been properly closed.

Can't Select Drive x, Status yy

This occurs for reasons similar to those given for the previous drive errors. However, this message usually indicates a hardware failure and displays the error status associated with the malfunction.

Can't Re-zero Drive x, Status yy

This error occurs when a hard disk drive cannot be rezeroed, or restored, without error. The associated error status is reported.

Init error: Drive x, Cylinder ww, Surface z, Status yy
Restore error: Drive x, Cylinder ww, Surface z, Status yy
Seek error: Drive x, Cylinder ww, Surface z, Status yy
Write error: Drive x, Cylinder ww, Surface z, Status yy

The error occurred on the specified drive, cylinder, and surface, which has the reported error status associated with it.

Formatting aborted just prior to writing cylinder ww, surface z

Although not an error, this message shows how much of the disk was formatted before the user pressed the ESCape key and aborted initialization.

PIO's not working
PIO's and direction control transceivers OK

The first of these is an error message displayed during the drive test phase of initialization if the PIOs on the WDI hard disk controller are not working correctly. The second message is displayed when the Init program sees the PIOs are working correctly.

Memory-to-memory DMA not working
Memory-to-memory DMA completed correctly

The Init program tests the hard disk DMA circuitry by using it to perform memory-to-memory DMA. The first message is displayed if the WDI hard disk controller DMA is not working correctly. The second, an informational message, is printed if DMA is working correctly as far as the Init program is able to determine.

No index pulses being received

This error message appears if the program receives no index pulses from the drive during the test phase of initialization. This indicates the drive is not rotating, or is not properly connected to the controller.

Index pulses being received correctly
Rotational speed: xxxx RPM

This informational message means that index pulses are being received from the drive and indicates the drive's rotational speed in revolutions per minute.

Rotational speed: overflow
Illegal drive speed (must be xxxx RPM +/- yy%)

These error messages mean that the rotational speed calculated by the Init program is out of the legal range for this type of drive. This is generally an indication of a drive malfunction.

ZPU clock must be set to 4MHz for correct operation of hard disk

This error message is printed if the operator is attempting to use a hard disk in a 2MHz (rather than a 4MHz) computer system. The problem can be corrected by switching the ZPU to 4MHz operation.

Incorrect operation of WDI and hard disk

This error message is displayed following any of several errors. It indicates a problem with the operation of either the hard disk or the WDI hard disk controller.

Read error: alternate track register
Do you wish to format drive anyway (Y/N)?

A list of alternate tracks for the hard disk (to be used in case of hard errors on any normal data tracks) is stored for use by the operating system in a dedicated area of track 0 (cylinder 0, surface 0). This area is called the alternate track table. Before initializing the drive, the Init program attempts to read this register. Those tracks that have already been declared are then redeclared after initialization.

This error message and question are displayed if the Init program is unable to read the alternate track table. Answering **no** to this question aborts the Init program until the problem is corrected. Answering **yes** to this question allows the Init program to go ahead and attempt to format the drive anyway. In this latter case, the alternate track table information is lost.

Cannot be assigned an alternate track

Since the alternate track table is stored on track 0 (cylinder 0, surface 0), this track can never be assigned an alternate track.

utility:	**INPUT**
purpose:	This program reads a line from the standard input and sends it to the standard output.
user access:	all users
summary:	input string
arguments:	none
options:	none

Description

This utility reads a string from the standard input and, upon reading a newline, sends that string to the standard output. This utility can be used to write interactive command language programs by redirecting the output of the utility to a file and then testing the contents of the file with the Testinp utility. Refer to the Testinp utility. Input reads a maximum of 255 characters from the standard input or 512 characters if input is redirected from a file. Input terminates reading upon reading a newline or the maximum number of characters, and does not output the newline to stdout.

Example:

```
% input > temp
```

This command line reads one line from the standard input and sends it to the file **temp**.

utility:	**IOPRUN**
purpose:	This program loads a program into an IOP.
user access:	privileged user
summary:	ioprun filename [address]
arguments:	filename of program to be loaded into IOP base address (in hex) of IOP (default CEh)
options:	none

Description

The Ioprun utility loads a file into an IOP. This utility is normally used to load the IOP/Quadart driver (**cromix.iop**) into the system IOP. Ioprun is found in the **/dev/iop** directory.

Shell command:	**KILL**
purpose:	This command sends a signal to a process.
user access:	all users
summary:	kill [-12345678] [PID]
arguments:	process id
options:	-1 abort
	-2 user
	-3 kill
	-4 terminate (default)
	-5 alarm
	-6 broken pipe
	-7 reserved
	-8 reserved

Description

The Kill command sends a signal to the process specified. If the signal type is unspecified, Kill sends a terminate signal. When a signal is sent to process 0, the signal is also sent to all processes belonging to that user.

If the user is a privileged user and a user signal is sent to process 1 (**kill -2 1**), system shutdown is initiated.

Kill 0 aborts all background jobs attached to the user's terminal.

Kill -1 1 consults the **/etc/ttys** file and allows any terminals that have been added to be logged on. It also logs off any terminals that have been deleted from the file.

Options

The -1 option causes an abort signal to be sent to the process. This option has the same effect as CNTRL-C from the keyboard, and aborts only interactive programs. Detached processes continue unaffected.

The **-2** option sends a user signal to the process. It is generated by a character typed at the terminal. The character that generates the signal is determined by the mode.

The **-3** option sends a kill signal to the process. This kill signal cannot be ignored or trapped. It is typically used to abort a program caught in an infinite loop.

The **-4** option sends a terminate signal to the process. The terminate signal kills both interactive and background processes. This is the default type of signal sent by the Kill command.

The **-5** option sends an alarm signal to the process.

The **-6** option is sent by the operating system when a pipe is used improperly.

The **-7** and **-8** options are reserved for future use.

utility:	L
purpose:	This program lists directory or file information.
user access:	all users
summary:	l [-abdeilrst] [file-list]
arguments:	optional file or directory pathname(s)
options:	-a all -b brief -d directory information -e everything -i inode number -l long list -r reverse order -s summary -t time modified

Description

The L program lists directory or file information in alphabetical order. If no pathname is specified, it lists the contents of the current directory. If a directory pathname is given, the contents of that directory are listed. If a file pathname is given, information about that file is listed.

Options

The **-a** option lists the names of all files, including invisible files (those files whose names begin with a period).

The **-b** option makes a brief list, which contains only filenames.

The **-d** option lists information about the directory, rather than the contents of the directory.

The **-e** option lists everything about a file.

The **-i** option lists an inode number, rather than the file size.

The **-l** option makes a long list of information. This option does not display as much information as the **-e** option.

The **-r** option performs the sort specified in reverse order. Thus, an alphabetical listing is given in reverse alphabetical order, and a time-date listing is listed most recent file first.

The **-s** option generates a summary of disk space used.

The **-t** option sorts the file list in order of time-last-modified. This order is from oldest to most recent unless the **-r** option is used.

Notes

The meaning of the first column of numbers displayed by the L utility is as follows. If the file listed is a regular (data) file, the number associated with the file is its size in bytes (or number of characters). If the file is a directory, the number is the number of files stored in that directory. If the file is a device file, the numbers are the major and minor device numbers.

Example:

Samples of the output from the L utility follow. Each is preceded by a note as to the option utilized.

The following shows an output of L with the **-b** option, containing only filenames.

```
apa
apal
apb
apc
apd
ape
```

The following shows an output of L using the **-d** option. For a filename, the field on the extreme left contains the number of bytes in the file. This is followed by the number of links to the file, and the filename. If the entry represents a directory, as in the first entry shown, the leftmost number shows the number of files in the directory. The **D** indicates it is a directory. The last two fields show the number of links and the directory name.

```
    3 D  1 cromix.doc
1,559    1 default.fm0
```

A sample of the output of **L** using the **-e** option is shown below. This is the most complete display. The name of each file in the directory is displayed on the extreme left. To the right, on the same line, is the number of bytes in the file. The first column on the next line lists the operations performed on the file: created, modified, accessed, or dumped. To the right of each operation is the date the operation was last performed. A third column shows the time of execution.

The rightmost column contains additional information. At the top the read, execute, write and access privileges for the owner, group, and all other users are shown. The second line is the login name of the file owner. The third entry lists the number of links to the file, and the final entry is the inode number.

To the extreme right of the owner's login name is an entry showing the group name of the user: in this case, **pubs1**.

```
Directory: cromix.doc
locktest                                   9  directory
        created:   Dec-21-1981 13:56:57  rewa re-- re--
        modified:  Dec-21-1981 13:56:57  karen          pubs1
        accessed:  Jan-19-1982 12:49:41  links: 1
        dumped:    000-00-1900 00:00:00  inode: 734

pipetest                                  10  directory
        created:   Dec-21-1981 13:56:13  rewa re-- re--
        modified:  Dec-21-1981 13:56:13  karen          pubs1
        accessed:  Jan-19-1982 12:49:33  links: 1
        dumped:    000-00-1900 00:00:00  inode: 781

system.c                               1,641
        created:   Dec-21-1981 13:56:10  rewa re-- re--
        modified:  Dec-21-1981 13:56:11  karen          pubs1
        accessed:  Dec-31-1981 12:17:05  links: 2
        dumped:    000-00-1900 00:00:00  inode: 782
```

The following example shows the L program output using the -i, or inode number, option. This display first shows the directory name. The names of all files and directories within the subject directory are listed on the right. The number of links to each file or directory is shown just to the left of the name. Moving left, the next field is either blank or contains a D. A

blank indicates the entry is a file; a D means it is a directory entry. The leftmost column in this display is the inode number associated with the file or directory.

```
Directory: cromix.doc
        734 D  1 locktest
        781 D  1 pipetest
        782    2 system.c
```

The following example shows the L program output using the **-l** option. If the second field in the entry is a **D**, for directory, the leftmost field indicates the number of files in that directory. If the second field is blank, the entry is a file, and the leftmost field shows the number of bytes in the file. Moving to the right, the third field indicates the number of links to the file or directory.

The next field shows the read, execute, write, and append access of the directory or file for the owner, group, and all other users, in that order. Immediately to the right of the access privileges is the login name of the owner. The three rightmost fields in this format are the most recent date and time of file access, and the file or directory name.

```
Directory: cromix.doc
       9 D  1 rewa re-- re-- karen        Dec-21 13:56  locktest
      10 D  1 rewa re-- re-- karen        Dec-21 13:56  pipetest
   1,641    2 rewa re-- re-- karen        Dec-21 13:56  system.c
```

The following is a sample of L program output using the **-s** option. This display is similar to that obtained using the **-d** option, except that the last line of the display is a summary showing, from left to right, the number of files, number of blocks, and total bytes in the directory.

```
Directory: cromix.doc
          9 D  1 locktest
         10 D  1 pipetest
      1,641    2 system.c
3 files    6 blocks       2,313 bytes
```

What follows is a sample of L program output using the -t option. These files are listed in order of the time last modified.

```
Directory: cromix.doc
       1,641     2 system.c
          10 D   1 pipetest
           9 D   1 locktest
```

utility:	**MAIL**
purpose:	This program sends or displays mail.
user access:	all users
summary:	mail [-agnvy] [user-name]
arguments:	optional list of user names **or** optional list of group names
options:	-a all -g group -n do not save mail -v verbose -y save mail

Description

Given without arguments, Mail displays mail sent to the user. After the mail is displayed, the Mail utility asks whether the user wants to save the mail. Saved mail is appended to the file **mbox** in the current directory.

Given with one or more user names as arguments, the Mail utility sends mail to one or more users. To send mail, enter the message after pressing RETURN at the end of the command line. A **CNTRL-Z** terminates the message and returns the user to the Cromix Operating System prompt. In order to send mail a user must have write and append access to the current directory, since mail creates a temporary file **mail.temp**.

Options

The **-a** option sends mail to all users. The list of users for this option is obtained from the **/etc/passwd** file.

The **-g** option sends mail to members of a specified group(s). Group members are defined in the **/etc/group** file.

The **-n** option causes mail not to be saved.

The **-v** option displays the list of users who received mail.

The **-y** option saves mail.

Notes

Upon logging in to the system, a user is informed if there is mail.

utility:	**MAKDEV**
purpose:	This program creates a device file.
user access:	all users
summary:	makdev [-c] devname b/c majornum minornum
arguments:	device name
	block or character device specification
	major device number
	minor device number
options:	-c conditional

Description

The Makdev utility associates a device driver with a number and a name. After the program is executed, references to the device name refer to the device indicated by the device number.

Options

The **-c** option displays an error message if no device driver corresponds to the specified device number.

Notes

Makdev calls for two numbers in its arguments: a major device number, which is the driver number, and a minor device number, which is the device number.

Some utilities demand that certain devices be owned by **bin.** For example, Spool expects the print devices to be owned by **bin.** Use the Chowner utility to change device ownership as needed.

Shell command:	**MAKDIR** or **MAKD**
purpose:	This command creates a directory.
user access:	all users
summary:	makdir dirl [... dirN]
arguments:	directory pathname(s)
options:	none

Description

The Makdir command creates directories.

utility:	**MAKFS**
purpose:	This program sets up the structure for a file system on disk.
user access:	privileged user
summary:	makfs [-ir #] devname
arguments:	device name
options:	-i number of inodes -r restore Superblock

Description

The Makfs utility sets up a structure for a file system on a block device. It establishes the number of inodes, the blocks dedicated to those inodes, blocks dedicated to the system, and blocks dedicated to the user.

Makfs is run on all floppy disks and on some hard disks before the disk is mounted for the first time.

The Makfs utility destroys any existing data on the device. It warns and prompts the user before destroying data.

The Makfs utility stores the inode number in all of the inodes created.

Options

The **-i** option establishes a file system with a nonstandard number of inodes. This option is used only if you need more files than the default allows. Otherwise, Makfs decides how many inodes are needed and uses that number.

The **-r** option restores the Superblock, should it be accidentally destroyed. This option should be used with caution. If you have an older version of the Makfs utility, using this option causes destruction of all data on the disk. After you have run Makfs -r, you must then run the Icheck utility to complete the restoration process.

Notes

A more prudent method of restoring the superblock is to use the Fixsb utility, which restores the Superblock and then runs Icheck automatically.

utility:	**MAKLINK**
purpose:	This program makes a link to a file.
user access:	all users
summary:	maklink [-fv] source-file destination-file [-fv] file-list dirname
arguments:	two single file pathnames **or** one or more file pathnames **and** a directory pathname
options:	-f force -v verbose

Description

The Maklink program links one or more files into a directory. This program does not alter the source file.

Options

The **-f** option causes the new link to overwrite another file with the same pathname if one exists. If the **-f** option is not used, and another file exists with the same name, an error results and Maklink is aborted.

The **-v** option displays the names of files as they are being linked.

Notes

No link is possible between two different file systems. That is, links cannot extend between two different devices (disks).

utility:	**MATCH**
purpose:	This program finds all occurrences of a string within a file.
user access:	all users
summary:	match [-bcelr] string file-list
arguments:	string file list
options:	-b block numbers -c count -e exact match -l line number -r reverse match

Description

The Match utility searches through the specified files for all occurrences of the string and displays each line containing a match. Unless the **-e** option is used, Match is not case sensitive. If no file is specified, input is accepted from the standard input device.

Options

The **-b** option displays the block number with the matching line.

The **-c** option prints a count of the matching lines. The lines themselves are not displayed.

The **-e** option displays only lines that match the given string exactly - a case sensitive match.

The **-l** option displays the line number together with the matching line.

The **-r** option reverses the sense of the match, displaying only lines that do **not** contain a match to the given string.

Notes

Strings of more than one word and ambiguous strings may be specified on the command line, surrounded by quotation marks. The same characters represent ambiguous strings as are used by the Cromix Shell (*, ?, []).

In addition, the caret character (^) may be specified at the beginning or end of a string to force the match of that string at the beginning or end of a line of text, respectively. The search for the string is case insensitive unless the -e option is used. If the ambiguous characters * or ? are used, the string should be enclosed in quotation marks (").

If match is used to search a file that is not a text file, control characters may be sent to the terminal. This may lock up the terminal; press CNTRL-Reset, or turn the terminal off and then on again to restore terminal operation.

Example:

```
% who
  john tty1
  roger tty2

% who|match roger

  roger tty2
```

utility:	**MODE**
purpose:	This program displays or alters the character device modes.
user access:	all users
summary:	mode [devname] [characteristic(s)]
arguments:	optional device name
	optional characteristic(s)
options:	-v verify

Description

The Mode utility program displays or alters the operational characteristics of a character device. If the program is run without any arguments, the current operational characteristics of the device from which the system received the Mode command are displayed.

To display the operational characteristics of another device, a device name must be specified as the first argument.

If no characteristics are specified, Mode displays the characteristics of the specified device without altering them.

Mode characteristics can be altered by specifying the desired settings as arguments. For example:

 # mode lpt1 width 132 -tabexpand

Some characteristics are switches that may be turned on or off. A dash is used to turn a switch off (e.g., **-tabexpand**). Omit the dash turn it on.

Some characteristics must be followed by numerical values, (e.g., **width 132**). Numerical values may be expressed as decimal or hexadecimal numbers, (e.g., **delaycode 7fh**). They may also be expressed by using the kilounit operator **K**, (e.g., **outblkln 8K**), where K = 1024.

Some characteristics use ASCII characters as values. Terminal devices have a line kill character which, by default, is CNTRL-U. Character values may be expressed

by either pressing the ASCII key itself, or by typing its hexadecimal value. For control characters, press the caret key (^) followed by the character. Thus, the line kill character can be changed to CNTRL-A by any one of the following methods:

1. Type **mode lkill**, then press the **A** key while holding down the **CNTRL** key;

2. Type **mode lkill 01h**;

3. Type **mode lkill^A.**

All commands are entered by pressing the RETURN key. Methods 2 and 3 are ways to make the RETURN key, for example, the user signal key. Either **mode sigchar 0dh** or **mode sigchar ^M** accomplishes this.

When displayed, the first part of the name of a mode characteristic is capitalized: **PAuse,** for instance. The capitalized part of the name must be used in changing the characteristic. For example, either **mode ttyl -pa** or **mode ttyl -pause,** but not **mode ttyl -p,** can be used to turn off the pause mode of ttyl.

Option

The **-v** option verifies Mode changes by displaying the characteristics after changing them.

Notes

In CBREAK mode, RAW mode, and BINARY mode, no calls for reading characters (.rdbyte, .rdline, nor .rdseq) wait for a line terminator; they all return after a single byte is entered.

The Shell, the program through which the Cromix Operating System reads command lines, sets the mode to nonCBREAK, nonRAW, and nonBINARY each time it prompts for a new command line. A program, PROG, can be run in BINARY mode, by typing

% **mode binary; prog**

3102 function keys are disabled and cause the terminal to **beep** when Cromix is initially booted up. They may be enabled with the Mode utility by using the command **mode fn**. The command **mode -fn** disables them again.

Enabling the keys allows the actual 2-byte sequence generated by the 3102 to be passed to a program. This sequence consists of a CNTRL-B (^B) followed by another character. For example, **CNTRL-B** and **p** are transmitted when function key 1 is pressed. These 2-byte sequences must then be intercepted by an application program to cause them to perform some command.

If no device is specified, the device from which the Mode utility was called is assumed to be the device in question. It normally defaults to the terminal calling the Mode utility. However, if the Mode utility is called from a command file, the disk drive where the command file is stored is considered the source device. In summary, if a command file is to change the mode of the terminal, the device must be stated explicitly.

A description of the mode characteristics for various kinds of devices follows.

TIMER (System Clock)

Correction

This is the number of seconds per 100 days to be added to or subtracted from the system timer. The range is -32768 to +32767. Only a privileged user may change this value. Refer to the section in Chapter 6, **Adjusting the System Clock**, for additional information.

TTY, QTTY, and MTTY (Terminal Devices)
LPT, TYP, SLPT, and QSLPT (Printer Devices)

ABortenable

This switch indicates whether or not CNTRL-C functions as a special input character for the terminal devices TTY, QTTY, and MTTY. When this switch is off, CNTRL-C will be treated as any standard character. When on, pressing CNTRL-C sends a SIGABORT signal. In order to disable the signal function of CNTRL-C, the user can give the command **mode -abortenable.** The argument **abortenable** enables the signal function of CNTRL-C.

Baud

This parameter determines the baud rate of the serial devices TTY, QTTY, MTTY, and SLPT. To change the baud rate, use the argument **baud** followed by the desired buad rate. For example:

```
mode tty5 b 9600
```

The baud rate designated **Auto** is a special case. This mode is used with a terminal and causes the driver to try different baud rates until it reads a RETURN from the input.

BINary, CBreak, and RAW

CBreak, RAW, and BINary are parameters of terminal devices TTY, QTTY, and MTTY. If any of these parameters is enabled, any read from the device returns after each input character. These parameters serve to disable the action of various other parameters. These effects are listed in the table below. (+ means that the parameter causes the given effect, a space means it does not.)

Effect	CBreak	RAW	BINary
Return after each character input	+	+	+
No erase, linekill, or EOF (CNTRL-Z) characters	+	+	+
No output PAuse or output Width truncation	+	+	+
Treat XOFF (CNTRL-S), XON (CNTRL-Q) as regular input		+	+
No tandem mode - no input buffer flow control			+
Treat CNTRL-C and SIGChar key as regular input			+
No checking or changing of input parity bit			+
No delays after any output control characters such as tabs			+
No echoing of input			+
No function key decoding			+
No character transformations - ignore the LCase, CRDEVice, and TABexpand settings		+	+

BMargin

If a printer device, LPT, TYP, SLPT, or QSLPT is within BMargin lines of the bottom of the page, a formfeed is generated. This takes the device to the top of the next page. The length of a page is determined by the parameter Length (see below).

CBreak

See BINary.

CRDEVice

This switch indicates whether or not the designated device is a carriage return device.

For a carriage return device, each RETURN character read from the device is translated into a newline character by the driver before being passed to the calling program. The driver then echoes a RETURN, linefeed sequence to the device. In the case of output to a carriage return device, newlines are translated into RETURN, linefeed sequences.

If a device is not a carriage return device, then it is a newline device and these translations are not made.

The hexadecimal code for the newline character is 0Ah, a single character that performs the function of a RETURN, linefeed sequence.

DELAYcode

The DELAYcode is the decimal equivalent of a byte determining the amount of delay inserted after certain characters are sent to the output. For TTYs, the delay is accomplished by sending null characters to the output. For QTTYs, the interrupt process is suspended for multiples of one-tenth of a second. Two bits on DELAYcode determine newline delay, 2 bits determine tab delay, 1 bit determines backspace delay, and so on. These bit assignments are as follows:

Character	**DELAYcode Bits**	**QTTY Values** (seconds)	**TTY Values** (nulls)
newline	0 and 1	0, .1, .2, .3	0, 4, 8, 12
tab	2 and 3	0, .1, .2, .3	0, 4, 8, 12
carriage return	4 and 5	0, .1, .2, .3	0, 4, 8, 12
formfeed	6	0, .8	0, 128
backspace	7	0, .1	0, 4

For example, **mode qttyl delaycode a3h** sets the QTTY1 newline delay to 0.3 seconds, the RETURN delay to 0.2 seconds, the backspace delay to 0.1 seconds, and the TAB and formfeed delays to zero.

DELECho

This is the character to be echoed in response to any one of the delete characters for terminal devices TTY, QTTY, and MTTY. When the Mode utility displays the delete echo character, the word DELECho is followed by the selected character. If the letter **R** appears in place of a single character, it indicates that a three-character sequence will be echoed in response to a delete character. This sequence is space backspace space.

To change the delete echo character, use the argument **DELECho** followed by a space and the delete character desired or the letter **R.**

DIScard

When a driver is first used, a data area is allocated where its parameters (including its mode characteristics) are saved. This data area is reserved for the driver until it is DIScarded. For most drivers, the location of the data area depends on the port address of the interface board used. For example, terminal TTY2 and serial line printer SLPT2 both use the TU-ART interface board addressed at 20h. For this reason, after access to TTY2 is obtained, SLPT2 cannot be opened until the driver for TTY2 has first been discarded. If the command **mode tty2 discard** is given, the data space for TTY2 is discarded as soon as the device TTY2 is closed. Then SLPT2 can be opened.

ECho

This switch determines whether characters entered on the terminal devices TTY, QTTY, and MTTY are echoed.

In order to disable character echo, use the argument **-echo.** To enable the echo, use the argument **echo.**

Erase

This controls the auxiliary erase character for terminal devices TTY, QTTY, and MTTY. The auxiliary erase character may be used to erase characters entered on the current line. In addition, there are always two standard erase characters. These are DEL (7Fh) and CNTRL-H (08h, also referred to as backspace).

To change the auxiliary erase character, use the argument **erase** followed by a space and the desired character. For example:

% **mode erase _**

This command line causes the underscore to function as an auxiliary delete character. Note that DEL and backspace still function as delete characters.

EVenparity

The two characteristics, ODDparity and EVenparity, produce four combinations. These are listed in the following table (where + means enabled and - means disabled).

EVenparity	ODDparity	Function for Input Characters
-	-	does not check parity but strips parity bit
+	-	checks for even parity before stripping parity bit
-	+	checks for odd parity before stripping parity bit
+	+	leaves parity unchecked and unchanged

EVenparity	ODDparity	Function for Output Characters
-	-	strips parity bit
+	-	makes character even parity
-	+	makes character odd parity
+	+	leaves parity bit unchanged

FFexpand

If FFexpand is on, every formfeed character (0bh) used as an output to printer devices LPT, TYP, SLPT, QSLPT is converted to newlines so that subsequent output starts at the top of the next page. The length of a page is determined by the parameter Length. If FFexpand is off, the formfeed character itself is an output to the device.

FNkeys

If FNkeys is enabled, the terminal drivers TTY, QTTY, and MTTY perform the handshaking that the Cromemco 3102 terminal expects whenever a function key is pressed. (The driver echoes a CNTRL-B for each of the two bytes the terminal sends.) This allows the 2-byte function key sequences of the 3102 to be transmitted to a program when a function key is pressed.

HUPenable

If this switch is on and an IOP terminal device, a QTTY or an MTTY, closes, the modem on the IOP device is hung up.

SIGHUPall

If this switch is on and the modem of an IOP terminal device, QTTY or MTTY, hangs up, the signal SIGHANGUP is sent to all processes controlled by the device. A process is controlled by the terminal with which the user who initiated the process logged in. For example, a user who has logged in on MTTY1 and hangs up without logging out is logged off by the resulting SIGHANGUP signal, provided SIGHUPall is enabled.

IMmediateecho

This determines the way that the terminal drivers TTY, QTTY, and MTTY treat type-ahead. If IMmediateecho is on, characters typed ahead are echoed immediately. They are echoed again when they are read.

If IMmediateecho is off, they will be echoed only at the time they are read.

Length

This is the page length in lines of the designated device. When the Mode utility displays the page length, the word length is followed by the specified page length. To change the page length, use the argument **length** followed by a space and the desired page length.

LCase

If LCase is on, terminal devices TTY, QTTY, and MTTY convert upper case alphabetic input characters to lower case.

LKill

The LKill character deletes the current input line for terminal drivers TTY, QTTY, and MTTY. This performs multiple deletes back to the last prompt character.

ODDparity

See EVENparity.

PAuse

If PAuse is on, terminal devices TTY, QTTY, and MTTY pause after of the number of lines specified by Length have been output. The output resumes only after an XON (CNTRL-Q) is entered on the keyboard.

RAW

See BINary.

SIGenable, SIGChar, and SIGALLchars

If SIGenable is on and SIGALLchars is off, pressing the SIGChar key causes terminal devices TTY, QTTY, and MTTY to send a SIGUSER signal to all processes controlled by the terminal. The SIGChar key character is not put into the input stream. If SIGenable is off, then the SIGChar key is treated in the same manner as any other key.

The terminal which controls a process is the terminal on which the owner of the process logged on to the system.

If SIGenable and SIGALLchars are both on, pressing the SIGChar key causes the SIGUSER signal to be sent to all processes controlled by the terminal, but the SIGChar key character is also put into the input stream.

If SIGALLchars is on but SIGenable is off, every terminal keystroke pressed before a system call to read input has been made sends the SIGUSER signal to all controlled processes. (Only characters typed-ahead send signals.) The characters are also put into the input stream.

Note that Shells are set up to ignore SIGUSER signals, so that a user is not logged off by them. Any program running in a nondetached mode that does not either ignore or trap SIGUSER signals is aborted by them. The .signal system call provides a means for ignoring or trapping signals.

TABexpand

If TABexpand is on, every tab character (09H) output is converted to enough spaces to bring the output to the next standard tab stop. Standard tab stops are multiples of 8 at columns 1, 9, 12, etc. on the terminal.

TANdem

Tandem mode is used to allow a receiving Cromix system to control the rate of input data using the DC1/DC3 handshaking protocol. The device sending data may be a Cromix system or another computer. When used to communicate between two Cromix systems, the ttys to be used in both the sending and receiving systems should **not** be selected in the ttys files. Both drivers should be conditioned to the same Baud rates, have RAW mode enabled, and ECho and CRdevice disabled. The receiving system should have TANdem mode enabled, and the receiving program or command file should already be executing before sending begins. Tandem mode causes the receiving system to transmit a DC3 (XOFF) character when the tty driver buffer is full. This causes the sending driver to stop sending. When the driver is ready to accept more characters, it transmits a DC1 (XON) character, and the sending driver resumes sending.

Width

The Width function specifies the number of columns displayed before truncation or wrap-around. If Width = 0, no truncation or wrap-around occurs.

WRAParound

If WRAParound is on, and the device output column reaches the page Width, an extra newline is sent to the device. This allows the remainder of the output line to be printed on the next line. If WRAParound is off, the remainder of the line is truncated. If Width = 0, no truncation or wrap-around occurs.

MODES FOR TP (Tape Devices)

Block

To move to a block within a tape file, use the argument **block** followed by a space and the block number. Tape blocks are numbered 1, 2, 3, and so on. The following example moves to the second block within the current tape file on device TP1:

```
% mode tp1 block 2
```

If the specified block is larger than the total number of blocks in the file, the device moves to the beginning of the next tape file.

BLKSwritten

BLKSwritten is a count of the blocks written when the tape file was last written to the tape device. It cannot be changed with the Mode utility.

EOFclose

If the EOFclose switch is on, a filemark is automatically written on the tape when the tape device is closed. A filemark marks the end of a tape file. If the switch is off, no filemark is written. A filemark is written on a tape when a .setmode system call is made for the tape device with the c register containing TPFMARK (0C6H).

File

To move to a file on tape, use the argument **file** followed by a space and the number of the file. Tape files are numbered 1, 2, 3, and so on. The following example moves to the sixth file on TP1:

```
% mode tp1 file 6
```

If the specified file number is larger than the total number of files recorded on the tape, the device moves to the end of the tape reel. This motion may be aborted by taking the tape drive off-line and pushing the

CNTRL-C key of the terminal keyboard. To take the drive off-line, push the ON-LINE button on the front of the tape drive until the ON-LINE light goes off.

Inblkln

Inblkln equals the length in bytes of the first block of the last file read from the tape device. It cannot be changed with the Mode utility.

Outblkln

Outblkln is the block length used by the driver writing files on a tape device. To set it, use the argument **outblkln** followed by a space and the desired size. The following command sets the output block length of tp3 to 8192 bytes, or 8K:

% **mode tp3 outblkln 8K**

REWind

To rewind a tape device, use the argument **rewind.** For example:

% **mode tp1 rewind**

UNLOAD

To unload a tape device, use the argument **unload.** For example:

% **mode tp2 unload**

utility: **MOUNT**
purpose: This program enables access to a file system.

user access: privileged users

summary: mount [-r] devname dummyname

arguments: device name

file pathname

options: -r read only

Description

The Mount utility enables access to a file system.

When given without any arguments, Mount lists the currently mounted devices.

The Mount utility looks on the disk to be mounted for the file **/etc/passwd.** Finding that file, it looks for the special user name **mount.** If this name is present and has a password associated with it, Mount prompts the user for the password before mounting the disk. Thus, it is possible to protect disks from being mounted by an unauthorized user.

Options

The **-r** option causes the file system to be mounted for read only access.

Notes

A file system that has been mounted **must be unmounted** by use of the Unmount utility **before the mounted disk is removed from the system.** If this is not done, the integrity of the data on the mounted system cannot be assured.

Do not attempt to mount a file system on a nonexistent device. Devices which do not exist may be deleted from the **/dev** directory.

Example:

```
% create newfilesys
% mount fdb newfilesys
% l
    .
    .
    .
145 D newfilesys

%
```

In the example above, the user first creates a dummy file. After mounting, the name of this dummy file becomes the root directory name of the file system to be mounted. After unmounting, this name becomes a dummy filename once again.

The Mount command is given with the device name where the file system is located. Refer to Appendix D for a complete list of device names.

The L utility shows that the new file system has been mounted and gives the name of the root directory.

utility:	**MOUNTHELP**
purpose:	Mounts the second 5-inch Cromix Operating System diskette.
user access:	privileged user
summary:	mounthelp
arguments:	none
option:	none

Description

The Mounthelp command mounts the second Cromix Operating System 5-inch diskette into the **/usr** directory. The Cromix Operating System can be used without mounting the second disk; the only difference is the online manual residing on the second disk will not be accessible.

utility:	**MOVE**
purpose:	This program moves file(s) from one directory into another.
user access:	all users
summary:	move [-ftv] file-list dirname [-ftv] srcfile destfile
arguments:	two single file pathnames **or** one or more file pathnames **and** a directory pathname
options:	-f force -t time -v verbose

Description

The Move program moves one or more files from one directory to another directory. This program destroys the source file(s). The Move program does not change the access privileges of the moved files. If files are transported from directory A to directory B, the owner of directory B may not have full access privileges for the files. The program Chowner must be run to change the owner of these files.

Options

The **-f** option causes the moved file to overwrite another file with the same pathname if one exists. If this option is not used and another file exists with the destination pathname, an error is generated and the Move program aborted.

The **-t** option causes a file to be moved **only** if:

1. The file does not exist in the destination directory; or

2. The source file was modified more recently than the destination file. This comparison is performed on a file-by-file basis.

The **-v** option displays the names of the files being moved.

utility:	**MSG**
purpose:	This program sends messages between users.
user access:	all users
summary:	msg [-any2] [user-name or devname]
arguments:	text terminated by CNTRL-Z
options:	-a all -n disable -y enable -2 Cromemco 3102 terminal

Description

The Msg utility sends messages between users or from a user to a device. Sending a message to a device is useful when a device is online but no user is in attendance.

If **msg** is typed and immediately followed by a RETURN, then a message is displayed to inform the user of the status of incoming messages. Incoming messages may be disabled or enabled by using the **-n** and **-y** options. Terminating a message with CNTRL-Z automatically sends the message **End of message** to the receiving user.

The Msg command followed by (optionally the -2 option and) a user or device name and RETURN allows a message to be entered. The message is transmitted to the destination user after each RETURN is pressed. A CNTRL-Z terminates the message and returns the originating user to the Shell.

Options

The **-a** option broadcasts a message to all users currently logged on to the system. This can be used by the privileged user to warn other users of interruptions to system usage such as rebooting. This message is sent to all users whether or not they have message receiving enabled. The message is preceded by the warning **Broadcast message.** Only privileged users are permitted to use this option. A message sent with the **-a** option

is not transmitted until the entire message is given. Hence, when the **-a** option is specified, it may be followed on the command line by the name of a file that contains a broadcast message.

The **-n** option causes incoming messages to be disabled.

The **-y** option allows incoming messages to be received.

The **-2** option sends messages to the status line of a Cromemco 3102 terminal.

Notes

To clear the status line of a Cromemco 3102 terminal after receiving a message transmitted using the -2 option, type **CNTRL-shift** followed by **CNTRL-1**.

If two-way communication is desired, a protocol should be established to prevent the confusion that arises when two messages are transmitted simultaneously. A suggested protocol follows: One user transmits at a time. A single **o** (short for over) is transmitted on a line by itself to indicate the end of the message. Upon seeing the **o**, the other user responds, terminating the message with an **o**. When the entire communication is finished, one user transmits **oo** (short for over and out) followed by a **CNTRL-Z**. The other user should type a **CNTRL-Z** also.

Two-way communication can be established by the Msg utility. When a user receives a message:

```
Message from xxxx
```

the receiving user should type:

```
msg xxxx
```

This allows users to send each other messages. In the example above, **xxxx** represents a user name.

utility:	**NCHECK**
purpose:	This program displays file information.
user access:	all users
summary:	ncheck [-i # # ...] [dirname or filename]
arguments:	directory or file pathname
options:	-i inodes

Description

The Ncheck program displays the inode number, link count, and pathname of all files contained in the specified directory and all subdirectories. If no arguments are supplied, Ncheck uses the root directory. The Ncheck utility obtains the inode number for a file from the inode itself.

Options

The **-i** option displays information about specified inodes only.

utility:	**NEWDISK**
purpose:	This program copies the system disk.
user access:	privileged users
summary:	newdisk devname
arguments:	device name

Description

The Newdisk utility copies the system disk. Newdisk must be followed by the name of the device on which the disk is to be created. The Newdisk command file first executes the Init program. Be sure to specify the correct disk drive, as data on the disk specified is destroyed in the copying process.

If you are updating your Cromix System disk to a new version of the operating system, refer to the Update utility.

utility: **NEWUSER**
purpose: This program displays information for new users.

user access: all users

summary: newuser

arguments: none

options: none

Description

The Newuser utility displays the file **newuser.msg**, which contains information about new or modified utilities, Shell commands, system calls, and other features of the present version of the Cromix Operating System.

utility:	**PASSWD**
purpose:	This program changes the passwd and group files.
user access:	all users
summary:	passwd [-dgn] [user1 user2...]
arguments:	user1 user2...
options:	-d delete -g group -n new user

Description

The Passwd utility has three functions. It may be used to change a user's own password. A privileged user may use it to add and delete from the list of users permitted to log on to the system. By using the delete function followed by the add function, the privileged user may change the login status of any user.

In any one of these three modes of operation, user name(s) are specified either on the command line or during the execution of the Passwd program.

To change the password only, enter the command **passwd** followed by a RETURN. The Passwd program prompts for a user name and a new password.

Options

The **-d** option deletes a specified user or group.

The **-g** option alters the **/etc/group** file (instead of the **/etc/passwd** file).

The **-n** option adds new user(s) or group(s).

Establishing a New User

A new user may be added using the Passwd program. In the following example, the user logs on as the privileged user **system** and creates a new user **fred** with the password **mountain:**

```
Login: system

Logged in system Jun-24-1980 17:12:15 on console
# passwd -n

Name: fred
Password: xxx
User number: 5
Group number: 0
Directory: /usr/fred
Starting Program:

Name:
#
```

The Passwd program prompts for a user name. The response to this prompt is the user name typed in response to the **Login:** prompt. Press RETURN after entering the name.

Next, the program prompts for a user password. If no password is desired, press RETURN in response to the prompt. If you do enter a password, it is encrypted, and the encrypted password displayed on the screen. When a user logs on, this password must be entered after the password prompt.

The program prompts for the user and group identification numbers. Each of these is an unsigned integer between 0 and 65535. A zero in the user field indicates a privileged user. A zero in the group field indicates the user is not a member of any group. Any other number has significance only within a given system.

The **Directory:** prompt allows specification of an initial directory, which is the user's home directory. If this directory does not exist, the system creates one. The user is the owner of this directory. If the home directory already exists, the Passwd utility prints this information.

Finally, the Passwd program prompts for a **Starting program.** If RETURN is pressed in response to the prompt, the user has full use of the Shell program. If the name of a program is entered here, the user is brought up running the program specified and is logged off upon exiting from the program. Any valid Shell command line may be entered in response to this prompt.

Deleting a User

A user is deleted from the list of users (**/etc/passwd** file) by running the Passwd program with the **-d** option. In the following example, the user **fred** is deleted:

```
# passwd -d

Name: fred

Name: RETURN
#
```

Note that only a privileged user may delete a user.

Changing a Password

When called without any options, the Passwd program allows the privileged user to change any user's password and any user to change his or her own password. To change a password, call the Passwd program as follows:

```
% passwd
Name: fred
Password: xxx

Name:  RETURN
%
```

Notice that the password encryption is displayed only after the password and a RETURN have been entered.

Changing User Characteristics

If the privileged user has occasion to change user characteristics other than the password, the user must be deleted and added again with the new characteristics.

utility:	**PATCH**
purpose:	This program patches a file.
user access:	all users
summary:	patch filename
arguments:	filename
options:	none

Description

This program displays and alters specified bytes within a file. Enter the command name plus a filename, and press RETURN. The program displays a greater-than sign (>). The user must enter one of three subcommands: **d** for display, **s** for substitute, and **e** for exit.

Notes

The **d** subcommand displays one sector of the file at a time, in a format similar to that used by the Dump utility.

The **s** subcommand displays the file word by word, so it can easily be changed.

The **e** subcommand allows you to exit from the program.

Shell command:	**PATH** or **PA**
purpose:	This command finds the full pathname of an executable file.
user access:	all users
summary:	path file-list
arguments:	filename
options:	none

Description

The Path command searches the current directory for the specified file with an extension of **.bin**, **.com**, or **.cmd**. It then searches the **/bin** directory for a **.bin** or **.com** file and the **/cmd** directory for a **.cmd** file. If the specified command is a Shell command, Path notifies the user of that fact. Path locates only executable files.

Path lets you make sure you are running the correct version of your program, rather than a copy that may have been altered.

Shell command:	**PRIORITY** or **PRI**
purpose:	This command changes the priority of a process.
user access:	all users (priorities 0 through +40) privileged user (priorities +40 through -40)
summary:	pri [±priority-number][command-line]
arguments:	priority number (optional) command line (optional)
options:	none

Description

The Priority command establishes the priority of a process. Priority numbers range from -40 (highest) to +40 (lowest). The highest priority a nonprivileged user may specify is 0, the lowest is +40. A privileged user may specify any priority.

If the Priority command is executed without a priority number, the default value is +10. All processes run without using the Priority command are assigned a priority of 0.

If a command line is given as an argument, the priority specified applies to the process(es) initiated by the command line. If no argument is given, the priority applies to the current Shell and all children of the current Shell created after execution of the Priority command.

utility: **PRIV**
purpose: This program allows any user the status of a privileged user.

user access: all users

summary: priv

arguments: none

options: none

Description

The Priv utility examines the **/etc/passwd** file for a user named **system.** If this user is not found, an error message is displayed and execution of the utility is aborted.

If the user named **system** is found and there is a password associated with the user, the Priv utility prompts for the password. If the user responds with the correct password or if no password is associated with the user **system**, a new Shell is formed in which the user has the status of a privileged user. Upon exiting from the new Shell, the user's previous status is reinstated.

Shell command:	**PROMPT**
purpose:	This command changes the prompt.
user access:	all users
summary:	prompt [char]
arguments:	any character
options:	none

Description

The Prompt command changes the prompt. **Char** is the new character that the Cromix Operating System is to use as a prompt. It must be a single character. If no character is specified, the prompt is changed to the pound sign (#) for the privileged user and to the percent sign (%) for any other user.

Notes

Changing the prompt from a percent sign to a pound sign does not make a user a privileged user.

Shell command:	**PSTAT** or **PS**
purpose:	This command displays the status of a process.
user access:	all users
summary:	pstat [-abl]
arguments:	none
options:	-a all -b brief display -l long display

Description

The Pstat command displays the following information on the status of a process:

PID	process identification number
state	state of process: Sleeping Ready Terminated
user id #	
group id #	
Ctty	controlling tty, the tty from which the process was started
Seconds	number of seconds the process has been executing
bank	memory bank in which the process resides
command line	command line which invoked the process

Options

The **-a** option lists the status of all processes. If the **-a** option is not selected, only those processes with the ID of the user giving the Pstat command are displayed.

The **-b** option displays a brief list of processes and their status.

The **-l** option displays a long list of processes and their status.

Example:

Below is a Pstat display. The first is in the long format, the second in the brief format.

```
PID State  UID   GID  Ctty   Pri  Bank  Seconds Command
  1   S      0    0   0:0     0    0     26.524  -
 36   R     23    1   1:0     0    0     36.676  shell
157   S     11    0   1:2     0    0     10.020  shell
253   S     21    1   1:5     0    0      4.460  shell
192   S     18    1   1:7     0    0     14.020  shell
290   S  32767    1   1:7     0    0      4.384  shell -p daemon /dev/typl
286   S     11    0   1:2     0    2     37.392  screen outline.txt
260   S     21    1   1:5     0    4     17.020  screen donahue
```

```
PID State Command
  1   S   -
 36   R   shell
157   S   shell
253   S   shell
192   S   shell
159   R   gtty 1 a tty4 03
290   S   shell -p daemon /dev/typl
286   S   screen outline.txt
260   S   screen donahue
```

utility:	**Q** or **QUERY**
purpose:	This program displays a short description of a specified utility program or Shell command.
user access:	all users
summary:	query [-s] [name]
arguments:	the names of one or more utility programs or Shell commands
options:	-s system function lists system call data as well as commands and utilities.

Description

The Query program searches a file containing one line descriptions of all of the utility programs and Shell commands for the name given as an argument.

When using Query without an argument, a listing of all one line descriptions of utilities and Shell commands is displayed.

The Query program considers names that are part of other keywords. When the name **fil** is given, Query finds all occurrences of the name **file** as well. This is helpful when the correct spelling of a name is unknown.

After using Query to find the name of the desired command, additional information is obtained by entering **help**, followed by the name of the command. For further details, refer to the Help utility.

The Query program uses the file **/usr/query/query_data** as a database. This file may be edited using the Screen Editor.

Options

The **-s** option searches the file **/usr/query/sys_data**, **/usr/query/jsys_data**, and **/usr/query/mode_data** before searching the default file, which gives information on the programs only.

The **/usr/query/sys_data** file gives a list of system calls associated with the command. The **/usr/query/jsys_data** and **/usr/query/mode_data** are linked to the files **/equ/jsysequ.z80** and **/equ/modeequ.z80**, respectively.

Example:

The following example demonstrates the use of the Query program.

```
% query delete

query_data
delete    - removes a file or directory from a file
            system
deltree   - deletes a directory and its descendents
passwd    - change a user password, add or
            delete a user
```

In the above example, the Query program has displayed all descriptions of Shell commands and utility programs that contain the word **delete** in their descriptions.

Shell command:	**RENAME** or **REN**
purpose:	This command changes the name and/or directory of a file.
user access:	all users
summary:	ren oldfile1 newfile1 [... oldfileN newfileN]
arguments:	one or more pairs of file pathnames (existing pathname first, followed by new pathname)
options:	none

Description

The Rename command changes a filename and/or the directory where it is located.

This command does not move a file from one device to another.

Shell
command: **REPEAT** or **REP**
purpose: This command repeats a command.

user access: all users

summary: rep count command

arguments: a count of the number of repetitions

command

options: none

Description

The Repeat command is used to repeat a command a specified number of times.

Example:

```
% repeat 5 echo "this line is displayed five times"
this line is displayed five times
this line is displayed five times
this line is displayed five times
this line is displayed five times
this line is displayed five times
%
```

Notes

The Repeat command may be terminated by a semicolon and in this case any command(s) following a semicolon are executed only once. This means that the following command displays the date three times and then displays the time once:

```
% repeat 3 date; time

Wednesday, November 12, 1980
Wednesday, November 12, 1980
Wednesday, November 12, 1980
Wednesday, November 12, 1980 18:54:04
```

utility: **RESTORE**
purpose: This utility restores data saved by the Backup utility.

user access: all users

summary: restore [-lv] source-dev [file-list]

arguments: source device

and

optional list of files to be restored

options:
-l list only
-v verbose

Description

The Restore program recreates files saved by the Backup program.

The Restore program always starts with the first disk (number 1) created by the Backup program and prompts the user as necessary for additional disks from the set of backup disks.

If no filenames are specified, the entire directory, including all descendant directories and files, is restored to its original structure. If one or more filenames are specified, the specified files are restored **into the current directory.** If a file list is specified, only files with names **exactly matching** those in the file list are restored.

Options

The **-l** option lists the names of all files backed up on the set disks. No files are transferred using this option. It may be used only with the first (number 1) disk in a set of backup disks.

The **-v** option displays the names of the files as they are restored.

Shell command:	**REWIND** or **REW**
purpose:	This command restores the arguments used to call a command file.
user access:	all users
summary:	rew
arguments:	none
options:	none

Description

The Rewind command restores the arguments used to call a command file. It nullifies the effect of any Shift commands given within the command file. After execution of the Rewind command, **#1** represents the first argument of the original command file, **#2** the second, and so on.

utility:	**ROOT**
purpose:	This program displays the name of the device containing the root directory.
user access:	all users
summary:	root
arguments:	none
options:	none

Description

The Root program displays the root directory's device pathname.

Example:

```
# root

  /dev/hd0
```

utility:	**RUNQD**
purpose:	This utility converts the Cromix Operating System for use with an IOP and Quadarts.
user access:	privileged user
summary:	runqd
arguments:	none
options:	none

Description

Runqd generates a version of the Cromix Operating System which uses an IOP and Quadarts as an interface. Refer to Chapter 6 for more information. The Runtu utility returns the operating system to its original set up.

utility:	**RUNTU**
purpose:	This utility reconfigures a system to run with TU-ARTs.
user access:	privileged user
summary:	runtu
arguments:	none
options:	none

Description

The Runtu utility reconfigures a system to run with TU-ARTs. This program effectively undoes the modifications made by the Runqd utility.

utility: **SCREEN**
purpose: This program is used to edit files.

user access: all users

summary: screen filename

arguments: name of file to be edited

options: none

Description

The Screen utility program enables the user to edit files. Please refer to the Cromemco Screen Editor Instruction Manual (part number 023-0081) for a complete discussion of the Screen Editor. This discussion covers those features of the Cromemco Cromix Screen Editor that are different from the Cromemco CDOS Screen Editor.

The Cromix Screen Editor is a special version of the Screen Editor, designed to take advantage of the features of the Cromix Operating System. It utilizes Cromix Operating System calls and does not use the CDOS Simulator. Because of this, full pathnames may be used when calling the Screen Editor.

The only difference which is apparent to the user is the addition of the % command. This command creates a Shell process, which allows the user to execute any commands provided there is enough memory in the system. Even without additional memory, any Shell command may be used. The user returns to the Screen program at any time by entering the Exit command in response to the Shell prompt.

Shell command:	**SHELL** or **SH**
purpose:	This command creates a Shell process.
user access:	all users
summary:	[shell] [cmd file]
arguments:	optional command file
option:	-c complete input line -p parsed input line -q quiet

Description

Given without an argument, the Shell command creates a Shell process. Given with the name of a file, the Shell command executes that command file. This can be useful if there are two files in the current directory with the same name, one having a filename extension of **bin**, the other **cmd**. If the file name is entered, the **bin** file is executed. If the Shell command is given with the **cmd** file, the command file is executed. In all other cases, the Shell command is implicit when the name of a command file is entered.

When a command file is executed by entering just the name of the command file (not preceded by the word **Shell**), the commands are not echoed to the console as they are executed. If the commands are to be echoed, the name of the command file should be preceded by the word **shell** on the command line.

Refer to Chapter 7, The Cromix Shell, for additional information.

Options

These options are needed only when a program is calling a Shell. They are not useful when a Shell is called from the terminal.

The **-c** option indicates that the command line being passed to the Shell is completed (i.e., has not been parsed).

The **-p** option indicates that the command line being passed to the Shell has been parsed.

The **-q** option requests that lines from a command file not be echoed to the terminal (standard output).

Shell command:	**SHIFT**
purpose:	This command shifts the arguments in a command file.
user access:	all users
summary:	shift
arguments:	none
options:	none

Description

The Shift command is used to shift the arguments in a command file. After execution of the Shift command, **#1** represents the second argument from the original command line, **#2** represents the third, and so on. After another execution of the Shift command, **#1** represents the third argument, etc.

The Rewind command nullifies the effects of the Shift command.

Example:

```
%abc
screen #1
shift
if "#1" != " "goto abc
```

If the command file above (named **abc.cmd**) is called as follows:

```
abc *.txt
```

the Shell expands the ambiguous filename ***.txt** into a list of all files in the current directory with the **txt** extension. The command file **abc** is then called with this list as an argument.

The first executable command within the command file is **screen.** Standard argument substitution causes replacement of **#1** by the first argument from the command line.

After **screen** is executed, the Shift command represents the second argument from the original command line **#1.**

This command line argument is substituted for **#1.** If this file exists, the string filename is not equal to the string " ", and control is transferred to the line labeled **abc.** If this file does not exist, a null string is substituted for **#1,** the string " " is equal to the string " ", and execution of the command file is terminated.

utility:	**SHUTDOWN**
purpose:	This program shuts down the system.
user access:	privileged user
summary:	shutdown
arguments:	none
options:	none

Description

The Shutdown command file contains commands to shut down the operating system by killing all processes, flushing buffers, and logging off all users. It first warns users and provides a 5-second countdown.

Shutdown also has a facility that works with the Startup command to detect inadvertent system terminations.

Run the Shutdown program whenever system operation is to be terminated.

utility:	**SIM**
purpose:	This utility allows CDOS programs to run under the Cromix Operating System.
user access:	all users
summary:	(sim) progname arg0,arg1,...,argn
arguments:	program name **and** arguments to the program to be run
options:	none

Description

The Sim program allows CDOS programs to run under the Cromix Operating System. The CDOS simulator is automatically loaded when a file with the extension **.com** is executed.

Notes

The Cdoscopy utility program is the only way to read files from or write files to CDOS format disks from the Cromix Operating System.

Drive/File Access From CDOS Programs

For CDOS programs to gain access to files on various drives, the CDOS Simulator converts disk specifiers to directory names. For example:

B:Filename becomes **/B/Filename**

If no disk specifier or the disk specifier **A** is used (as in **A:Filename**), the file is assumed to be in the current directory.

To take full advantage of this scheme, Cromemco recommends a file structure be constructed as follows:

1. Create files **B**, **C**, **D**, etc. in the root directory. Each file corresponds to one of the disk drives in the system.

2. Mount each disk on the appropriate drive using the Mount utility:

   # mount fdb /b

   Note that these **must be** Cromix format disks.

3. The files on those disks may be read and written from CDOS programs. The CDOS Simulator, running under the Cromix Operating System, automatically converts the CDOS drive specifiers to the appropriate directory names.

4. Each disk mounted must be unmounted before it is physically removed from the system. To do this, use the Unmount utility:

   # unmount fdb

Disks created in this manner are in the Cromix Operating System format and not CDOS compatible.

Shell command:	**SLEEP** or **SL**
purpose:	This command suspends program execution.
user access:	all users
summary:	sleep time
arguments:	time in seconds
options:	none

Description

The Sleep command suspends execution of a process for the number of seconds specified. Sleep can be used to execute a command after a certain amount of time. For example:

```
sleep 60 ; command
```

This example executes the command after 60 seconds, or one minute. The time specified must be less than 65,536 seconds.

utility:	**SORT**
purpose:	This utility sorts or merges files.
user access:	all users
summary:	sort[-bdfirmut][±x.y][+pos][-pos] ... [-o name]filename(s)
arguments:	input filename(s)
options:	-b leading spaces and tabs ignored
	-d dictionary order
	-f consider upper case as lower case
	-i ignore all control characters and 7Fh in sort keys
	-r reverse order
	-m merge sorted input files only
	-o output file
	-u unique records only
	-t? use ? as field separator
	±x.y sort on keys

Description

The Sort utility has many options, each of which is described at length below. Before discussing these, the basic, or default, version of Sort is described.

Sort arranges the lines in a file or files in ASCII order. ASCII order puts nonprinting characters first, followed by blanks, punctuation, digits, upper case and lower case alphabetic characters. The ASCII table is shown in Appendix E of this manual.

Each line in a file is a record. A line is a string of characters terminated by a newline (0Ah). When Sort is used without the +x.y option, it sorts on the entire record. White space (blanks and tabs) separates fields within a record. Each field starts with a space or tab.

Where no input or output file is specified, the input is assumed to be the standard input device, and output is sent to the standard output device.

Options

The **-b** option causes leading tabs and spaces to be ignored. Lines or fields are sorted according to their first nonblank character. The resulting output integrates lines or fields having leading blanks with the other lines or fields in the file. Figure 9-1 shows a sample input file containing leading blanks. Figure 9-2 represents the output when the **-b** option is **not** used, and Figure 9-3 represents the output generated using the **-b** option.

```
maser
McCormack
MacDowell
          McKinley
mace
MacLeish
make
```

Figure 9-1: SAMPLE INPUT FILE WITH LEADING BLANKS

```
          McKinley
MacDowell
McCormack
MacLeish
mace
make
maser
```

Figure 9-2: SAMPLE OUTPUT USING NO OPTIONS

```
MacDowell
MacLeish
McCormack
          McKinley
mace
make
maser
```

Figure 9-3: SAMPLE OUTPUT USING -b OPTION

Figure 9-2 shows the output of a sort without the **-b** option. The record containing the most white space comes first in order, since blank spaces precede characters and letters in an ASCII sort. In Figure 9-3, the records are in ASCII order: alphabetized, but having upper case entries first. Note that the blanks and tabs are retained in the sorted output, even though they are not considered in the ordering of the file.

The **-d** option sorts the lines of the file in dictionary order. Dictionary order means that only letters, numbers, and blanks are considered when ordering the input file. This option discounts nonprinting characters and special characters that precede alphanumerics in ASCII order.

Figure 9-4 shows a file containing special characters, as well as alphanumerics. This file, **db.in**, when used as input for a sort without options, results in the sorted output shown in Figure 9-5.

```
a

+++
aaa
BBBBB
C
+C
BBB
CC
. 444
**a
**b
aAa
AAA
```

Figure 9-4: INPUT FILE db.in

The following sort statement generates an output file, **db2.out**, that does not use the **-d** option:

% **sort -o db2.out db.in**

The output is shown below:

```
**a
**b
+++
+C
. 444
AAA
BBB
BBBBB
C
CC
a
aAa
aaa
```

Figure 9-5: OUTPUT OF SORT OF db.in (NO OPTIONS)

Sort used without a **-d** option puts lines starting with special characters, such as the asterisk (*), before lines that start with capital letters. This is a standard ASCII sort. Figure 9-6 is the same file after being sorted using the **-d** option.

The sort statement:

```
% sort -d -o db.out db.in
```

produces a file in dictionary order.

```
+++
. 444
AAA
BBB
BBBBB
+C
C
CC
**a
a
aAa
aaa
**b
```

Figure 9-6: SORTED OUTPUT FILE IN DICTIONARY ORDER

The file in Figure 9-6 is alphabetically ordered, upper case letters first. If a line contains only special characters, the program resorts to ASCII order and lists it first. Where a line contains both special and alphabetic characters, special characters are treated as blanks.

Notice the output of a **dictionary sort** is not in simple alphabetical order. In this **dictionary sort**, upper case letters precede lower case letters.

The **-f** option considers upper case letters as lower case letters for the purpose of comparison. Special and nonprinting characters retain their order of precedence.

Figure 9-7 shows the sorted output of **db.in** (Figure 9-4) using the **-f** option.

```
**a
**b
+++
+C
. 444
a
AAA
aAa
aaa
BBB
BBBBB
C
CC
```

Figure 9-7: SORT OUTPUT USING THE -f OPTION

Note this sort comparison respects the precedence of special characters but ignores upper and lower case.

The **-r** option reverses the sort order, so that a **z** precedes an **a** in a sorted list. The input shown in Figure 9-8 is in random order. A sort of this file using the **-r** option produces the file shown in Figure 9-9. The sort statement is

% sort -r -o m.out m.in

```
cat
hat
mat
flat
bat
scat
sat
pat
    knat
    splat
    slat
  1
  !
Fatty
Vat
```

Figure 9-8: RANDOM INPUT FILE

```
scat
sat
pat
mat
hat
flat
cat
bat
Vat
Fatty
  1
  !
    knat
    splat
    slat
```

Figure 9-9: SORTED OUTPUT USING -r OPTION

Notice the **-r** option causes a complete reversal of the ASCII ordering scheme for blank space, letters, and numbers.

The **-m** option merges previously sorted input files. The merge option interleaves the records of each file, creating an ordered output only where input files are sorted according to the same scheme.

Figure 9-10 shows the contents of a sample input file, sorted without using options, which is to be merged with itself.

```
Bat
Fat
Hat
at
cat
rat
```

Figure 9-10: FILE ml.in

To obtain a merged output file, the sort statement is given as follows:

% **sort -m -o ml.out ml.in ml.in**

The result is shown in Figure 9-11, below.

```
Bat
Bat
Fat
Fat
Hat
Hat
at
at
cat
cat
rat
rat
```

Figure 9-11: MERGED OUTPUT OF TWO ASCII SORTED FILES

In this case, the merged output is in ASCII order, as were the input files.

It is possible to merge more than two files: the actual limit is 29, provided there is adequate disk space to support the files and a work area.

The **-u** option deletes the duplicate records in a file, leaving one copy of each record. For instance, the output of the merge operation, **ml.out**, should result in an output identical to **ml.in** when sorted using the **-u** option. The sort statement reads as follows:

```
% sort -u -o u.out ml.out
```

The result, shown in Figure 9-14, is as predicted.

```
Bat
Fat
Hat
at
cat
rat
```

Figure 9-14: OUTPUT OF THE SORT OPTION -u

For the **-u** option to identify two records as a match, they must be identical in all respects. Unless you specify differently, the **-u** option takes capitalization, blank space, and punctuation into consideration.

The **-t** option substitutes the character immediately following the option statement for space and tab field separators. This option allows use of any character as a field delimiter. The implications of this option are illustrated using a file called **fiz.in**, shown in Figure 9-15.

```
field
fzield
fizld
Field
fiezld
fieLzd
fieldz
```

Figure 9-15: INPUT FILE fiz.in

Figure 9-16 shows the output when **fiz.in** is sorted without options. This output file is a simple ASCII sort.

```
Field
fieLzd
field
fieldz
fiezld
fizld
fzield
```

Figure 9-16: ASCII SORT OF FILE fiz.in

To use the letter **z** as a field delimiter in sorting the file, insert the **-t** option in the sort statement and specify z as the delimiter:

% sort -tz -o fiz.out fiz.in

The result, shown in Figure 9-17, is arranged very differently than the ASCII sort of the file shown in Figure 9-16. The **-t** option caused a change in record order by varying the field length.

```
Field
fzield
fizld
fiezld
fieLzd
field
fieldz
```

Figure 9-17: fiz.in SORTED USING Z AS FILE DELIMITER

The **+x.y** option causes the input file to be sorted by keys. The option specifies the place on the line where the key (or field) to be sorted on starts. When using this option, **x** is replaced by the number of fields to be skipped; **y** is replaced by the number of characters to skip within the selected field. If **x.y** is specified as positive, the fields are skipped from the beginning of the line; if **x.y** is negative, the fields skipped are counted from the end of the line. When x=0, it indicates the first field on the line; x=-0 indicates the last field on the line.

For example, Figure 9-18 shows an input file **alpha** containing four records, each of which is grouped into eight fields of 3 bytes each.

```
abc def ghi jkl mno pqr stu vwx
Abc dEf ghI Jkl Mno pQr sTu Vwx
aBC DeF gHI Jkl mNo Pqr StU vWx
ABc dEF GHi jKL mnO pQR sTu VWx
```

Figure 9-18: INPUT FILE ALPHA

To sort the file using the second field in the record, **def**, the sort statement is written as follows:

% sort -d +1.0 -o Alpha.out Alpha

This statement sorts the file in dictionary order, using the second field in each record. The output file is shown in Figure 9-19. Notice the field is sorted starting with the first character in that field.

```
aBC DeF gHI Jkl mNo Pqr StU vWx
ABc dEF GHi jKL mnO pQR sTu VWx
Abc dEf ghI Jkl Mno pQr sTu Vwx
abc def ghi jkl mno pqr stu vwx
```

Figure 9-19: OUTPUT FILE alpha.out

To sort this file using only the second and third characters in the field, the sort statement is written

% sort +1.1d -O Alpha.out Alpha

The second letter in the second field is used to sort the file in dictionary order. The **d** option here follows the field to which it pertains.

To take a more complex example, the file **who** contains information on immigrants to California. The five fields in each record are as follows:

Last name
First name
Country of Origin
Street
County

Suppose we wish to sort the file first by country of origin. Within each national group, we then sort by last name, then first name. The final sort is by county. In the surname field, it is desirable to ignore upper and lower case letters.

The input file is:

Lopez	Jack	Spain	Bower	Orange
McNiff	John	England	Rose	Sonoma
Rizzo	Jill	Italy	Bly	Kings
Ross	Jerry	Wales	Green	Placer
Mcniff	John	England	Greer	Placer

Figure 9-20: FILE who

Each of these fields is separated by a tab, rather than blanks. The **-t** option is used in the sort statement to make the tab (CNTRL-I) the field separator. The sort statement is as follows:

sort -ft^I +2.0 +0.0 +1.0 -0.0 -o who.out who

The final field designation, -0.0, could instead have been stated as +4.0, had we wished to count fields from left to right. All these fields are sorted starting with the first element in the field. Figure 9-21 shows the output file, **who.out**.

Mcniff	John	England	Greer	Placer
McNiff	John	England	Rose	Sonoma
Rizzo	Jill	Italy	Bly	Kings
Lopez	Jack	Spain	Bower	Orange
Ross	Jerry	Wales	Green	Placer

Figure 9-21: FILE who.out

As indicated by the **-f** option in the sort statement, the difference in capitalization between Mcniff and McNiff was ignored, and the two records were ordered on the basis of the contents of the county field.

utility:	**SPOOL**
purpose:	This utility queues files and sends them to a printer.
user access:	all users
summary:	spool[-adhklqv][-c#][-m#][-p#][devname]pathname(s)

arguments:

device name

If no device name is specified, output is directed to **/dev/prt.** The device name may be used to direct the output of the Spool program to any of the system's printers.

pathname

Filenames must be used to add files to the printing queue. Filenames or the sequence numbers assigned by the Spool program may be used to delete or change priority.

options:

Adding files

-d	enter and delete
-h	header
-m	multiple copies
-p	priority
-v	verbose
RETURN	message

Listing files

-l	list
-la	list all

Changing priority

-c	change priority

Deleting files

-k	kill
-q	quit
-qa	quit all

Description

The Spool utility allows one or more users to send printing jobs to one or more printers in an orderly sequence that may be changed at any time.

If no file is specified, input is taken from the standard input device. This means the Spool utility can be used with redirected input or pipes.

When the Cromix Spool utility is called to add files to the printing queue, the files are copied into a directory named **/usr/spool.** The execution of the Spool utility requires one bank of user memory.

After the execution of the Spool program with any of its options, the specified files are sent to the printer **without use of any user memory.** This is accomplished by a function intrinsic to the Cromix Operating System.

Output from the Spool program may be directed to any character device located in the device table (**/dev**).

If no device is specified, **/dev/prt** is assumed. The Cromix Operating System is shipped assuming a dot matrix printer as the system printer. If a different printer is to be used as the system printer, refer to Chapter 6 to change the type of printer.

As requests are made to print additional files, the Spool program forms a **print queue.** Each file entered into the queue is assigned a unique **sequence number.** Once in the printing queue, files may be referenced by their filename or sequence number.

If two or more files in the queue have the same filename, a reference to that filename refers to all files with the same name. For example, if the **k** (kill) option is used with a filename that appears more than once in the queue, all files with that name are deleted from the queue. The sequence number can always be used to refer to a specific file.

Each file added to the printing queue is assigned a priority number ranging from 0 to 9. Zero is the highest priority and is reserved for a privileged user. If no priority is specified, a value of 5 is assigned automatically. A priority number must be specified when using the change priority option.

If two users request a print job with the same priority, the requests are serviced on a first come, first served basis.

A user other than a privileged user has access only to files which that the user placed in the printing queue. The priority of a file in the printing queue can be changed by the user who initiated the printing request or by a privileged user. In a similar manner, only the privileged user or the user who added a file to the printing queue can delete the file from the queue by using the kill option. Any user can list **all** of the files in the printing queue by using the **la** (list all) option.

Ambiguous file references must be used with caution. When an ambiguous file reference is expanded, it generates a list of filenames matching files in the current directory. An ambiguous file reference can be used when giving the Spool program files to add to the printing queue.

Ambiguous filenames are expanded from a directory, and not from a spool queue. An ambiguous file reference does not work properly when killing or changing the priority of files in the printing queue if files of the same name as in the spool queue do not exist in the current directory. This is the case when the delete option is used as files are added to the printing queue, or if the current directory is changed by the user.

If Spool is interrupted for any reason such as a power failure, jobs are left in the queue. There are three methods to restart Spool. Before restarting Spool, the printer should be manually brought to top-of-form.

The first method is to spool another job. This restarts the spool at the beginning of the first job in the queue (the job that was interrupted).

The second method, used when there are no more jobs to be spooled, is to enter the command line:

 # daemon /dev/yyy

where yyy is the device name of the printer being spooled to (usually **prt**). This also restarts the spool at the beginning of the interrupted print job.

The third method is to delete all spool jobs using the command line:

 # spool -qa

and then respool all unprinted jobs.

Options for Adding Files

The **-d** option adds all specified files to the spool queue and deletes them from the directory in which they reside. This option may include a device name, and must include a list of one or more filenames.

The **-h** option causes all specified files to be preceded by a one page header. The first line of the header page contains the name of the user who spooled the file, the date and time, and the name of the file. This is followed by the same information displayed in large characters. The large character portion of the header page truncates the user and filenames to eight characters. Note that the header uses the full width of standard 132 column paper.

The **-m** option prints files a specified number of times. The maximum number of copies is 255.

The **-p** option assigns a priority number to a printing job at the time it is initiated. The option must be followed by the desired priority number and may include a device name.

The **-v** option displays the list of files being processed. It may be used with all options except list and message.

The RETURN option allows the user to place a message in the printing queue. To do this, type the program name **spool** followed immediately by a RETURN. Enter the desired message terminated by **CNTRL-Z**. This option may include a device name, and allows the Spool utility to use redirected input.

Options for Listing Files

The **-l** option lists all jobs in the printing queue that the user initiated. They are listed in a table with the following information:

1. **Filename** of print file
2. Name of **user** requesting printing job
3. **Sequence** number of printing job
4. Destination **device** of printing job
5. **Priority** of printing job
6. **Pages** in printing job
7. **Lines** in printing job, and
8. **Copies** to be printed.

A privileged user always gets a list of all jobs in the printing queue.

The **-la** option lists **all** printing jobs in a table. Refer to the list option.

Options for Changing Priority

The **-c** option sets the priority of all specified files in the spool queue to the specified value. This option is followed by a priority number, and must include a list of one or more filenames or sequence numbers.

Options for Removing Files from the Spool Queue

The **-k** option deletes all specified files from the spool queue. If a specified file is printing, the printing is aborted. This option must include a list of one or more filenames or sequence numbers.

The **-q** option deletes all files which have been directed to the specified device from the spool queue.

The **-qa** option may be exercised only by a privileged user. It deletes **all** files that have been directed to the specified device from the spool queue.

Notes

Where no option is specified, the files specified by the pathname are added to the printing queue. A device name may be specified.

If more than one option is used, and one or more of the options requires an argument, the following syntax should be followed.

```
% spool -hv -m 3 -p 1 filename
```

The options that do not require arguments (**h** and **v** above) are grouped, preceded by a dash (-), and followed by a space. This group is followed by the option(s) which require arguments. Each option is preceded by a dash and followed by a space, a number, and another space. Additional option and argument pairs may follow. Finally, the filename(s) of the file(s) to be spooled are entered.

In the following examples, assume the print files **t**, **u**, **w**, **x**, **y**, and **z** exist in the current directory. First, place each of these files in the printing queue:

```
% spool -v t u w x y z
t
u
w
x
y
z
%
```

Because the verbose option is used, the Spool program listed each file as it was copied to the **spool** directory. The list option is then used to display the printing queue:

```
spool -l

   Filename   User  Seq  Device    Pri  Pages  Lines  Copies
-> t          fred  36   5:5 prt   5    2       95    1
   u          fred  37   5:5 prt   5    2      107    1
   w          fred  38   5:5 prt   5    1       42    1
   x          fred  39   5:5 prt   5    2      115    1
   y          fred  40   5:5 prt   5    2      115    1
   z          fred  41   5:5 prt   5    3      160    1
```

The arrow at the upper left of the listing indicates the file currently being printed. All jobs have a priority of five because no priority was indicated when the jobs were put in the queue.

Next, change the priority of file **y** to 2 and change the priority of the file with the sequence number 39 (file **x**) to 3. Then delete the file **u** from the queue using the -k option. Finally, add a message to the printing queue using the message option, and display the revised printing queue.

```
% spool -c 2 y
% spool -c 3 39
% spool -k u
% spool
this is a message
^Z% spool -l

   Filename   User  Seq  Device    Pri  Pages  Lines  Copies
-> t          fred  36   5:5 prt   5    2       95    1
   y          fred  40   5:5 prt   2    2      115    1
   x          fred  39   5:5 prt   3    2      115    1
   w          fred  38   5:5 prt   5    1       42    1
   z          fred  41   5:5 prt   5    3      160    1
   ----       fred  42   5:5 prt   5    1        2    1
```

Remember a message must be terminated by a **CNTRL-Z**, which echoes to the console as ^Z.

To spool multiple copies of a job, the **-m** option is used.

Example:

The command

% spool -m 3 pay7000

prints 3 copies of the report **pay7000.**

The command

% spool -hm 3 pay7000

prints 3 copies of **pay7000** with one header page at the beginning of each copy.

A pipe can be used to redirect output from a program to the printer. The following command line generates a list of the current directory on the printer.

% l | spool

utility:	**STARTUP**
purpose:	This file contains commands that are executed whenever the system is started up.
user access:	all users
summary:	startup
arguments:	none
options:	none

Description

The **startup.cmd** file resides in the **/etc** directory. As shipped, the command file contains a command to execute the time program that sets the system clock and date.

After the system is booted, Startup notices whether the system was last shutdown properly. If it was not, Startup informs the user the check program should be run to verify file system integrity.

Shell command:	**TEE**
user access:	all users
summary:	tee pathname
argument:	pathname
options:	none

Description

Tee takes input from the standard input file and sends it to the standard output, as well as to the file specified by the pathname provided in the argument.

Example:

% sort short | tee sort0

This command sorts the file **short,** and sends the sorted output to the terminal (standard output) and to the file named **sort0** in the current directory.

utility:	**TESTINP**
purpose:	This program tests the contents of a file for a particular string.
user access:	all users
summary:	testinp [-dfr] file string1 [string2 ...]
arguments:	file pathname one or more strings
options:	-d deletes -f compares first characters -r reverse sense of test

Description

This utility compares the contents of a file to a string or strings and sets an error return code if one of the strings does not match the contents of the file specified.

The test made by Testinp is case insensitive; a test string can be in upper, lower, or mixed case and matches a string that is in upper, lower, or mixed case.

Options

The **-r** option reverses the sense of Testinp by setting the error code if a match **does** occur.

The -f option checks only the first character of the file passed as an argument against the first character of each of the control strings.

The -d option deletes the file passed as an argument after the test. This option is useful in many command files using a temporary file created during the command file execution.

Example:

```
echo "Do you want to shut down the system?"
input > temp
testinp -d temp YES OUI SI
if -err goto noshutdown
kill -2 1
%noshutdown
```

The example above is a typical command file that uses Testinp and Input. The first line sends the string within quotation marks to the standard output. The second line uses the Input utility to send the user's response to the file **temp.** On the third line, Testinp tests the contents of the file **temp** for occurrences of the strings YES, OUI, or SI. Testinp then deletes **temp.** If the file contains one of the control strings, the system is shut down using the Kill command. If the file **temp** does not contain one of the control strings, Testinp sets an error code. The command that follows passes control to the label noshutdown. If the user answers **no** to the question, the system is not shut down.

utility:	**TIME**
purpose:	This program displays or alters the time and date.
user access:	all users for display privileged user for changes
summary:	time [-sde2t] [hour:minute:second] [month/day/year]
arguments:	optional date optional time
options:	-s set -d date only -t time only -2 see 3102 -e European style display (dd/mm/yy) -s2 set 3102 clock

Description

The Time program displays or changes the time and date. If no arguments are given, the current date and time are displayed. If the **-s** option is used, the user is prompted for the date and then the time. Although the date is displayed with the / separator, and time is displayed using the : separator, any convenient separator character (such as a space or a period) can be used when entering the date and time.

Options

The **-s** option causes the user to be prompted for a new date and time.

The **-e** option causes the time to be displayed in the European style, with the day and month reversed.

The **-d** option allows the user to set the date only. It is often used with the **-s** option.

The **-t** option allows the user to set the time only. It is often used with the **-s** option.

The **-2** option allows the user to retrieve and display the time stored in the 3102 terminal of the user making

the request. This option may be used to set the time on the user's 3102 terminal, if used in conjunction with the **-s** option. This option may be used to set the time of each user's terminal. The system time is maintained internally and is used for all system functions (such as the times associated with file creates, modifications, and dumps.)

Notes

The 3102 clock may be set by the **-s2** option, but it is not utilized by the Cromix Operating System.

The date should precede the time if both are given. If they are not supplied and the **-s** option is given, the Time utility prompts the user for them.

Example:

% time -s 03/26/82 14:30:24

% time -sd 03/26/82

% time -st 14:30:24

Shell command:	**TYPE** or **TY**
purpose:	This command displays a file in ASCII.
user access:	all users
summary:	ty [file-list]
arguments:	optional file pathnames
options:	none

Description

The Type command displays the file(s) specified by the pathname(s). Type may be used only to display ASCII (text) files. The reader is referred to the Dump utility for information on displaying other types of files.

Type uses stdin if no file list is given, and the output is sent to stdout. This means that type may be piped to or from, and redirected to or from, as shown in the example.

Example:

 # ty /dev/qtty5 > diskfile

This command line accepts data from **/dev/qtty5** and sends it to **diskfile.**

utility:	**UNMOUNT**
purpose:	This program disconnects a mounted file system from the current file system.
user access:	privileged user
summary:	unmount devname [-x]
arguments:	device name
options:	-x do not eject disk

Description

The Unmount utility program disables access to a file system. A file system that has been mounted **must be unmounted** by use of the Unmount utility **before the mounted disk is removed from the system or the system is powered-down.** If this is not done, the integrity of the data on the mounted system cannot be assured.

Options

The **-x** option causes a floppy disk not to be ejected when it is unmounted.

utility:	**UPDATE**
purpose:	This command file updates a Cromix system disk with a newer system disk.
user access:	privileged user
summary:	update devname
arguments:	device name
options:	none

Description

The Update program updates a Cromix system disk with a newer Cromix system disk.

utility:	**USAGE**
purpose:	This program displays directory size information.
user access:	all users
summary:	usage [file-list]
arguments:	directory or file pathname(s)
options:	none

Description

The Usage utility displays the physical disk space (in blocks) and the logical file space (in bytes) occupied by a directory and all of its descendant directories and files. If only a single file is specified, the size of that file is reported. If no pathname is given, the current directory is assumed.

Knowing the number of blocks occupied by a directory is useful when using the Cptree utility.

utility:	**VERSION**
purpose:	This program displays the version number of the Cromix Operating System or a utility program.
user access:	all users
summary:	version [file-list]
arguments:	optional file and/or directory list
options:	-v verbose

Description

When called without any argument, the Version utility displays the version of the Cromix Operating System being run. When called with the name of a utility program, Version displays the software release number of that utility. When called with a directory name, Version displays the version number of each of the programs in the directory. The following command displays the version numbers of all of the programs in the **/bin** directory:

% **version /bin**

The characters **RB** appearing in an entry indicate that the file is a Relocatable Binary file. This type of file may share a bank of memory with another process.

Options

The **-v** option causes the pathnames of files to be printed.

Shell command:	**WAIT**
purpose:	This command suspends execution and waits for the PID specified process to terminate.
user access:	all users
summary:	wait [PID]
arguments:	optional PID number
options:	none

Description

The Wait command causes the Cromix Operating System to suspend operation until the process specified by the process id number (PID) has terminated. If no process is specified, Wait suspends execution of the current process until all detached processes belonging to that user have terminated.

utility: **WBOOT**
purpose: This program initializes the boot track of a floppy disk.

user access: all users

summary: wboot devname

arguments: device name

options: none

Description

The Wboot utility writes the contents of the **/etc/fdboot** (large floppy disk) or **/etc/sfdboot** (small floppy disk) to the boot track of the disk in the specified device.

utility:	**WHO**
purpose:	This program displays a list of users who are currently logged in.
user access:	all users
summary:	who [/etc/account] [am i]
arguments:	optional /etc/account **or** optional am i
options:	none

Description

When the Who utility is called without an argument, the **/etc/who** file is consulted and a report is displayed showing the users currently logged on, together with the time each one logged on.

When followed by **am i**, the name of the user calling the Who utility is displayed.

If the Who utility is called followed by **/etc/account**, the information contained in the account file is displayed.

Chapter 10

SYSTEM CALLS

Calls to the Cromix Operating System are formed using a Z-80 restart instruction (RST 8) followed by a byte specifying the system call number.

The Cromemco Macro Assembler (version 03.07 and higher) contains an opcode (JSYS) that forms these two bytes in the object code. JSYS takes the Cromix system call number as its only operand. The files **jsysequ.z80** and **modeequ.z80** are provided to facilitate programming system calls. These files contain EQUates for all of the system call numbers and mode options so that the calls may be made by name and the numbers need not be remembered. To make use of these files, include them in the source file using the ***include** statement of the assembler.

For example:

```
*include jsysequ.z80
*include modeequ.z80

        jsys     .create      ;system call to create
                              ;and open a file



        ld      b,stdin       ;standard input channel
        ld      c,MD_ISPEED   ;input baud rate
        ld      d,S_2400      ;set to 2400 baud
        jsys    .setmode      ;system call to set
                              ;the specified mode
```

All system calls require the specified calling parameters. In addition, some calls return parameters. Parameters are passed to and returned from the system calls in registers or register pairs. All registers not specified as containing a returned parameter are preserved through a system call.

The following list summarizes the Cromix Operating System calls described in detail in this chapter.

SUMMARY OF SYSTEM CALLS

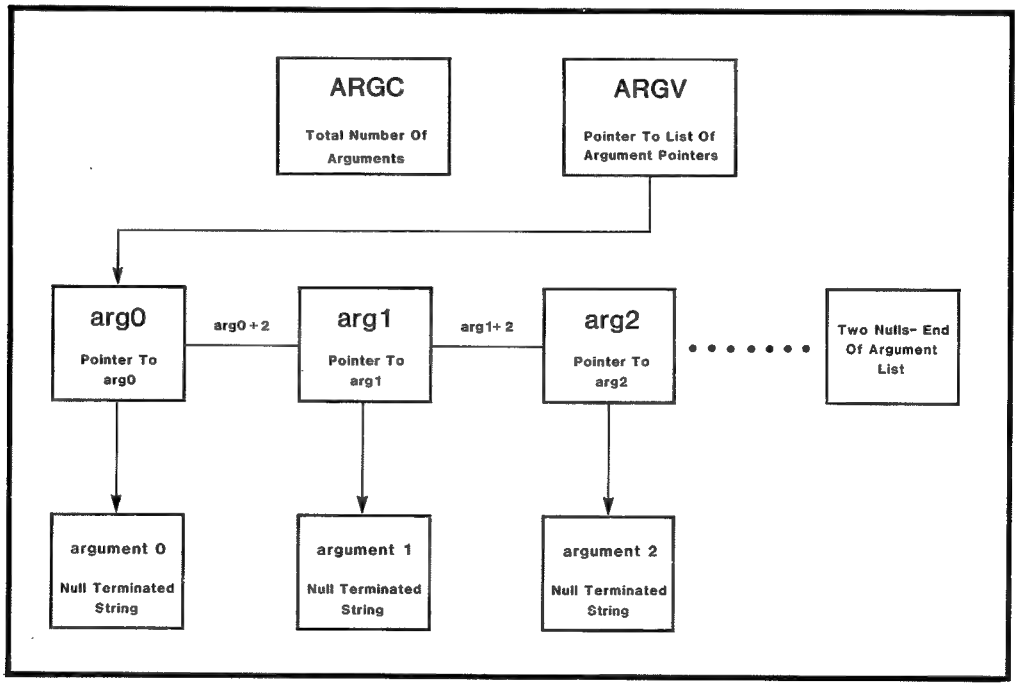
.alarm	sends an alarm signal to the current process after a specified number of seconds
.caccess	tests channel access
.cchstat	changes access privileges
.chdup	duplicates a channel
.chkdev	checks for the presence of a device driver
.clink	establishes a link to an open file
.close	closes a file
.create	creates and opens a new file
.cstat	determines the status of an open file
.delete	deletes a directory entry
.divd	divides one number by another
.error	displays an error message
.exchg	exchanges the data pointer fields of two inodes
.exec	executes a program
.exit	exits from a process
.faccess	tests file access
.fchstat	changes the status of a file
.fexec	forks and executes a program
.flink	establishes a link to a file
.fshell	forks a Shell process
.fstat	determines the status of a file
.getdate	returns the current date from the system clock
.getdir	returns the pathname of the current directory
.getgroup	returns the group id

.getmode	returns the characteristics of a character device
.getpos	returns the logical position of the file pointer
.getprior	returns the priority number of the current process
.getproc	returns the process id of the caller's active process
.gettime	returns the current time of the system clock
.getuser	returns the user id
.indirect	executes the system call
.kill	sends a signal to a process
.lock	assists in locking records
.makdev	creates a new name for a device
.makdir	creates a new directory
.mount	enables access to a file system
.mult	multiplies one number by another
.open	opens a file for access
.pause	suspends execution and waits for a signal
.pipe	creates an input and output channel for interprocess communication
.printf	outputs a formatted string to a specified file
.rdbyte	reads the next sequential byte from an open file
.rdline	reads a line
.rdseq	reads the next specified number of bytes
.setdate	changes the Cromix clock to the specified date
.setdir	changes the current directory
.setgroup	changes the group id of the current process

.setmode	changes the characteristics of a character device
.setpos	changes the file position pointer to the place specified
.setprior	changes the current process priority
.settime	changes the system clock to the specified time
.setuser	changes the user id
.shell	transfers execution to a Shell process
.signal	sets up to receive a signal
.sleep	puts a process to sleep for a specified number of seconds
.trunc	truncates an open file
.unlock	assists in unlocking records
.unmount	disables access to a file
.update	updates all current files with the contents of their buffers
.version	returns the operating system version number
.wait	waits for termination of a child process
.wrbyte	writes a byte to the specified file
.wrline	writes a line to the specified file
.wrseq	writes sequentially to the specified file

COMMAND LINE ARGUMENT PROCESSING

The Cromix Shell allows the user to pass arguments from the command line to a program or command file. Two labels have been included in the **jsysequ.z80** file to simplify argument processing. These labels are **ARGC** and **ARGV.** ARGC is a pointer to the ARGument Counter and ARGV is a pointer to the ARGument Vector. The argument counter contains the total number of command line arguments entered. This includes the program or command name (if no arguments are passed, ARGC = 1). ARGV is a pointer to a list of pointers, each of which points to one of the arguments. Each argument is a null terminated string. The argument list is terminated by two null characters.



The following memory dump displays all of the pointers involved in the command line **debug test.** If you examine location 40 hex in the user's bank (ARGC), you find the number 2. This is the total number of arguments that were passed (**debug+test**). Location 42h (ARGV) contains the least significant byte of the argument pointer list (E6h). Location 43h contains the most significant byte of the argument pointer list (FBh). These memory locations form the address FBE6h, which is the address of the argument pointer list. Examining location FBE6h,

you find the number ECh. This is the least significant byte of the address of the first argument. Location FBE7h is the most significant byte of the address of the first argument (FBh). These memory locations form the address FBECh which is the starting address of the string **/bin/debug.com** (the operating system expands the program name to its full pathname). The next pointer location, FBE8h (FBE6+2), contains the pointer to the second argument (**test**). The next pointer in the list is FBEAh. This location contains two nulls; this is the end of the argument pointer list.

```
0000   C3 03 EE FF  FF C3 00 CB  C3 F5 FD FF  FF FF FF FF  C.n..C.KCu}.....
0010   FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ................
0020   FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ................
0030   00 00 00 FF  FF FF FF FF  C3 0A FE FF  FF FF FF FF  ........C.~.....
0040   02 00 E6 FB  FF 00 FF 00  FF 00 FF 00  FF 00 FF 00  ..f{............
0050   FF  .

FBE6   EC FB FB FB  00 00 2F 62  69 6E 2F 64  65 62 75 67  l{{{../bin/debug
FBF6   2E 63 6F 6D  00 74 65 73  74 00 FF FF  FF FF FF FF  .com.test.......
FC06   FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ................
```

The following program listing is an example of the usage of ARGC and ARGV. The purpose of the program is to display all of the arguments passed to it on the console. In addition, it displays the argument number of each argument.

Example:

```
;
;       This program prints all arguments passed to it on the console
;
;
*include jsysequ.z80                            ; standard equates for Cromix

arg_test:       ld      sp,stack                ; load the stack pointer

                ld      ix,(argv)               ; get argument list vector pointer
                ld      a,(argc)                ; get the argument counter
                ld      e,a                     ; move it to a work register
                ld      c,0                     ; zero argument number
                                                ; (used by printf)

loop:           ld      l,(ix+0)                ; move the pointer to HL
                ld      h,(ix+1)                ; /

                ld      b,0                     ; zero b for use with printf
                push    hl                      ; print the argument pointer
                push    bc                      ; and the argument number
                ld      b,stdout                ; on the console
                ld      hl,ctrl_string          ; using the printf system call
                jsys    .printf                 ; /
                pop     bc                      ; restore the stack
                pop     hl                      ; /

                inc     ix                      ; point to the next argument
                inc     ix                      ; / pointer
                inc     c                       ; update the argument number

                dec     e                       ; update the argument counter

                jr      nz,loop                 ; if another argument
                                                ;       print it
                                                ; else
                ld      hl,0                    ;       termination status
                                                ;       = no errors
                jsys    .exit                   ;       exit to the operating
                                                ;       system

; printf control string and stack area

ctrl_string:    defb    'argument %d = %s\n',0

                defs    10h                     ; program stack
stack:          equ     $                       ; /

                end     arg_test
```

SIGNALS

A signal carries messages between processes. There are eight types of signals that can effect eight different responses from a process. The programmer can choose any one of three responses to each of seven of the eight types of signals. The **SIGKILL** signal in all cases, causes a process to be aborted.

Responses to a Signal

When a process receives a signal, the signal can be handled in one of three ways.

1. **Ignore the signal.**
   The process continues as though no signal had been received.

2. **Abort the process.**
   The operating system terminates the process. This is equivalent to execution of the **.exit** system call.

3. **Transfer control.**
   A user program may establish a location to which control may be transferred for each type of signal received.

After a signal has been received, the **.signal** system call must be executed again in order to be able to receive the next signal.

Types of Signals

The eight types of signals are enumerated below.

1. **sigabort**
   This is the abort signal generated by a CNTRL-C typed at the terminal. The mode of the terminal must be set to allow CNTRL-C to function (abortenable).

2. **siguser**
   This is the user signal generated by a character typed at the terminal. The character that generates this signal is determined and enabled by mode (sigcharacter and sigenable).

3. **sigkill**
This is the kill signal. It cannot be ignored or redirected by the user program. The kill signal causes the operating system to abort the process immediately. The kill signal can only be sent to a process by the initiator of the process or a privileged user.

4. **sigterm**
This is the terminate signal. It is the default type of signal for the Kill command of the Shell.

5. **sigalarm**
This is the alarm signal. It is sent by the operating system following an **.alarm** system call.

6. **sigpipe**
This is the pipe signal. It is sent by the operating system when a pipe is not being used properly.

7. **sighangup**
This is a signal sent by the mtty device when the phone hangs up, if the HUPENABLE mode is set.

8. reserved for future use.

Sources of Signals

Signals may be sent to a process by a user typed character, the Kill command, the **.kill** system call, or the **.alarm** system call.

Reception of Signals

A process may be set up to receive and process a signal by the **.signal** system call. If the signal is not ignored and the process has an unsatisfied request for input or output from a character device such as a terminal or printer, the input or output request is canceled.

A child process may be set up by its parent process to ignore or be aborted by a signal when the parent initiates the child through the **.fexec** or **.fshell** system call.

The bits of the B and C registers are used to specify the child's responses to each of the 8 signals. The signals to be ignored or to abort the child process are masked by setting the bits (=1) in the B register

corresponding to the signal. The bit in the C register corresponding to the signal is set (=1) if the signal is to be ignored by the child or reset (=0) if the signal is to abort the child.

The parent may pass along its own responses to signals to the child process, by resetting (=0) the bit in the B register corresponding to the signal. If the parent's response is provided by a routine at a transfer address, the child will be aborted by the signal.

The .signal system call sets up a process to receive a signal. The type of signal to be received is loaded in the C register. The execution address is loaded in the HL register. This is the address to which control is passed once the signal is received. The previous execution address is returned in the HL register pair.

A process which is run as a detached job by the Shell (through the use of & on the command line) is set up by the Shell to ignore **sigabort** and be aborted by **sigterm.** A process which runs in the forground (not detached) is set up by the Shell to be aborted by **sigabort** and ignore **sigterm.** These features allow the user to abort the current process by entering CNTRL-C, while not affecting detached processes and allow implementation of the Shell command **kill 0.**

The .kill system call sends signals to processes. The identification number of the process to which the signal goes is loaded in the HL register pair. The number of the signal type sent is loaded in the C register. A user may only send a signal to a process which that user initiated. Only a privileged user may send signals to processes initiated by other users. When a signal is sent to process 0, that signal is sent to all processes belonging to the user who invoked the call. If a privileged user sends SIGUSER to process 1, system shutdown is initiated. When SIGABORT is sent to process 1, the Cromix system consults the **/etc/ttys** file to log on any terminals that have been added and log off any deleted terminals.

The Use of Signals in Application Programs

The **.signal** system call is commonly used to catch or ignore CNTRL-C (sigabort) or other signals. A typical example is a text editor. An editor must catch or ignore CNTRL-C, entered by the user, to avoid possible disaster when the editor is terminated in the midst of file modification. By loading the HL register pair with 1 before any **.signal** system call is made, the programmer

causes the signal to be ignored. To cause the system to perform a specific function on receiving a CNTRL-C, the programmer loads the HL register with an address to which execution passes when the signal is received.

Immediately after a signal is received, the process is automatically set up to ignore further signals. If the process is to receive and handle any further signals, the **.signal** system call must be repeated.

If the HL register pair is loaded with 0000 before a **.signal** system call is made, execution of the process will be aborted when a signal of the type specified in the C register is received. If the **.signal** system call is not sent, the process is aborted when any signal is received.

Signals have many uses, but they also have limitations. Signals are designed to terminate processes or wake them up. Signals are not interrupts. Signals can be ignored, but not disabled. Mutual exclusion cannot be easily achieved with signals. If an application requires that a process receive and process several signals per second from one or more processes, difficulties with stack overflow are likely to arise.

Two sample uses for signals are included on the following pages. The first is a program fragment that catches the sigabort signal sent by a CNTRL-C entered on the keyboard. This might be useful in a program such as an editor in which program termination by a CNTRL-C could cause data loss. The second is a more complicated example that reveals the complexities involved in using signals within certain applications.

```
;       Program fragment demonstrating the Signal system call
;       used to catch a signal
;       SIGSETUP - Set up to receive a SIGABORT signal (CNTRL-C)

sigsetup:
        ld      hl,abort_vector         ; Address of routine to handle CNTRL-C
        ld      c,sigabort              ; Load c with signal type to catch
        jsys    .signal                 ; Make Cromix signal system call
        jp      c,error                 ; If error then jump to error routine
                                        ; Else continue
        ret                             ; Return

;       Note: the following routine can be located anywhere in the program
;       ABORT_VECTOR - Location where control is to pass after receiving a
;                      sigabort signal.
abort_vector:
        ld      hl,message              ; Load address of message string
        ld      b,stdout                ; The standard output channel
        jsys    .printf                 ; Print the message on the console
        jp      c,error                 ; If error jump to error routine
        ret                             ; Return to the location in code
                                        ; that was interrupted by the signal

message:defb    "Don't use CNTRL-C to quit \n",0
                                        ; A typical message to be printed
                                        ; upon receipt of a abort signal

error:  ld      b,stderr                ; Channel for error messages
        jsys    .error                  ; Call Cromix to write the error msg.
        ld      hl,-1                   ; Set error code
        jsys    .exit                   ; Exit to the operating system
```

The following is an example which provides a detailed description of the function of signals in the Cromix Operating System. The example is written in C language. This example is an implementation of the standard C function **system.** This function executes the command contained in the character string **s,** and then resumes execution of the current program. The order of events is as follows:

1. Set SIGABORT and SIGTERM signals to the ignore state, saving their previous values.

2. Fork the intended command string to the operating system, saving the process identification number of the forked process.

3. Call the **.wait** system call to suspend execution of the current process until the forked process has terminated.

4. Restore the status of the SIGABORT and SIGTERM signals to their previous values.

5. Check the termination status of the forked process. If it was terminated by either SIGTERM or SIGABORT, pass those signals on to the current process with the use of the **.kill** system call.

Note that when the process is forked, **.fshell** is called with the signal mask set to be the same as the parent process. The process identification number of the forked process is saved after the **.fshell** system call. This allows use of the **.wait** system call to halt execution of the current process until the child process has terminated. Upon its termination, the **.wait** system call returns and puts the termination status of the process into the variable STATUS.

The **.signal** system call is made to restore the values of the original signals. We check the status variable to find out if the child process was killed by either SIGTERM or SIGABORT. If so, the appropriate signal is passed to the parent with the **.kill** system call. Unless the parent had previously used the signal system call to set up for receiving that signal, the **.kill** system call kills the parent process.

This final step allows the child process to pass an abort signal back to the parent process.

```
/*
 *      Title:          SYSTEM(S)
 *      Description:    Execute the command contained in the character string S
 */

#include <jsysequ.h>

system(str_ptr)
char *str_ptr;
{
  struct command {                        /* Structure for fshell system call */
    char *sh ;
    char *cc ;
    char *string;
  } ;
  struct command  com = {"shell","-c",""};

  int oldtval, oldaval;                   /* Variables to store the previous   */
                                          /* TERMINATE and ABORT signal masks */
  int status[2];                          /* The status returned by WAIT       */
  int pid;                                /* Process ID of forked process      */
  com.string = str_ptr;                   /* Put command string in structure   */
/*
Step 1: Set up to receive sigabort and sigterm signals
*/
  oldaval = signal(1,sigabort);           /* 1, means ignore the signal        */
  oldtval = signal(1,sigterm);
/*
Step 2: Call FSHELL to fork the process.
*/
  pid = fshell(com,0xB,0)                 /* Save the child process id in pid */
/*
Step 3: Call WAIT to wait for the forked process to terminate
*/
  wait(0,pid ,status);
  /*
   *     Now restore the old signal values
   */
  signal(oldtval,sigterm);
  signal(oldaval,sigabort);

  /*
   *     Finally we check the return status that the forked process returned
   *     on completion. If it was killed by SIGABORT or SIGTERM then we pass
   *     that same signal on to the current process
   */

  if ( status[1] == sigabort)             /* If process was killed by SIGABORT */
    kill(getproc(),sigabort);             /* ...then send sigabort signal to   */
                                          /* current process.                  */
  if ( status[1] == sigterm )             /* Do the same if killed by SIGTERM  */
    kill(getproc(),sigterm);
}
```

The Alarm System Call

After a specified number of seconds, the **.alarm** system call sends an alarm signal (SIGALARM) to the process that made the system call. The **.signal** system call is first used to set up the process for receiving the SIGALARM signal. A typical use of **.alarm** provides a time out feature for a program. If a process must be prevented from hanging on an input request indefinitely, the process first makes the **.alarm** system call. The **.alarm** system call specifies the number of seconds to wait after making the request for input.

The Pause System Call

The **.pause** system call is frequently used in conjunction with the **.alarm** system call. The **.pause** call suspends execution of the calling process and waits for a SIGALARM signal. The **.pause** call does not require the **.signal** system call to set up the process to receive the signal. It is ideal for putting a process to sleep until another process signals it to continue. The **.pause** and **.alarm** calls can be used together to put a process to sleep for a specified number of seconds. For example:

```
sleep10:ld      hl,10           ; Send Alarm in 10 seconds
        jsys    .alarm          ; Call Cromix
        jp      c,error         ; If error then jump to error routine
        jsys    .pause          ; Wait for alarm signal
        jp      c,error         ; If error then jump to error routine
```

The Sleep System Call

The equivalent of the routine above can be achieved with one system call, **.sleep**. The **.sleep** call stops execution of a process for a specified number of seconds. The result shown above can be accomplished as follows using **.sleep**.

```
sleep10:ld      hl,10           ; Set up to go to sleep for 10 seconds
        jsys    .sleep          ; Call Cromix
        jp      c,error         ; If error then jump to error routine
```

Record Level Locks

The **.lock** system call assists in implementing record level file locks, and allows the operating system to absorb part of the overhead involved in the procedure. No locks are imposed by the operating system; this is done by the application program. The **.lock** and **.unlock** calls merely make and delete entries in a table residing in the system memory bank.

The **.lock** system call enters a string in the lock table. This string is the unique identifier of a record in a file. The string is hereinafter referred to as the **lock sequence.** Should another process make a **.lock** system call using a lock sequence currently in the lock table, the Cromix Operating System does one of two things. It either puts the process to sleep until the entry is removed, or it returns with an error code set. An entry is removed from the table when the process that made the original **.lock** system call reverses it with an **.unlock** system call, followed by the same lock sequence. Any process put to sleep while attempting to lock that sequence is awakened and allowed to make an entry in the table.

The problem of record level lock is resolved by preceding any read or write to a file or record with a **.lock** system call. This achieves mutual exclusion for records and avoids the undesirable effects of having multiple processes reading and writing the same file or record.

The other considerations associated with the **.lock** system call are the type of lock to be made and the character string to be used as the lock sequence.

Shared and Unshared Locks

A shared lock allows other processes access to the lock. Shared locks are typically used when a file is being read. A shared lock does not prevent other processes from entering the file, so that a process that is reading a record does not prevent another process from reading the file. A process attempting to establish an unshared lock when a shared lock has been granted either is put to sleep or receives an error.

Unshared locks are typically used during a write to a file, since they prevent any other process from getting access to the lock sequence. If a process has an unshared lock, any other process attempting to lock the same sequence either is put to sleep or receives an error.

Conditional and Unconditional Locks

Locks can be made conditionally or unconditionally. A conditional lock returns with an error code set if the sequence specified cannot be locked. An unconditional lock puts the calling process to sleep if the sequence is currently locked. The process put to sleep awakens when the process that originally issued the **.lock** call issues an **.unlock** call.

The programmer must decide whether to use a conditional or unconditional lock. For many applications, putting a process to sleep for a brief period because another process has locked a file or record does no harm. In other cases, such a maneuver may suspend execution of a program indefinitely while waiting for some process to unlock a file or record. In this case, a conditional lock may be used to print an error code informing the user that the record or file is in use. An ideal strategy might employ both techniques, or use the **.alarm** system call to prevent indefinite postponement of file access.

Locking Schemes

If more than one program is relying on the **.lock** system call, a mutually agreed upon scheme must be devised so that all programs use the same identifier to reference records in a file. This identifier is the locking sequence and may contain from one to 16 bytes. An example of a locking sequence is the first 8 bytes of the filename followed by the number of the record to be locked. This scheme works as long as no two files simultaneously in use have names beginning with the same eight characters, and as long as two different processes do not access the same file through two links having different names.

A more elaborate locking scheme uses the file device and inode numbers. The combination of device and inode numbers is a unique file identifier. The number of the device on which a file resides can be obtained by using

the **.fstat** system call. The locking sequence could be composed of a device number followed by an inode number and a record number.

If the number of available locks is exceeded, the operating system returns from a **.lock** system call with an error message. This message merely indicates there is no room left in the lock table.

A **?deadlock** error is returned if the operating system detects a deadlock condition.

All locks installed by a process are automatically unlocked when the process is terminated.

Sample Implementations of Locks

The uses of record locks are best shown through illustration. Consider an inventory management system on a multi-user Cromix system at a music store. If salesperson A sells a guitar and wishes to decrement the inventory record, the program would enter a section of code designed to perform the following functions:

1. Request record number to read.

2. Lock the record with a shared, unconditional lock.

3. Read the record.

4. Unlock the record.

The program might then inform the salesperson that three guitars are in stock. The salesperson rings up the sale, decrements the count of guitars in stock to two, and writes the record to the database using an unshared conditional lock during the write. Difficulties arise if another salesperson, B, also sells a guitar at the same time. B might read the record at the same time as A, decrement the inventory, and write the file out to the database. The record shows that two guitars are in stock, when in fact, there is now only one.

There are several possible solutions to the problem. The simplest is to make an unshared lock at the time of the original read and perform the unlock only after the record had been written out. The problem with this scheme is the potential for barring another user from access to the record for a long time.

A more adequate solution to the problem is to let the system resolve possible conflicts. All user reads are preceded by a shared lock, which permits simultaneous access of the record by other users. When the modified record is to be written out, the system checks to see if the record has been modified in the interim period. If it has not been changed, it is written out. If it has been changed, the value of the record must be recalculated.

EXECUTING MORE THAN ONE PROCESS IN A BANK

The Cromix Operating System supports execution of more than one process in a single user bank of memory. The Blink utility optionally generates files in a relocatable binary (RB) format. RB files are capable of sharing a bank of user memory with other RB programs or with a single non-RB program.

A user bank of memory is always in one of three states.

1. It is completely empty (not in use).

2. It is partially used.

   a. It is in use by one or more RB programs, but still has room left to execute another program (partially used).

   or

   b. It can be in use by a regular program that began execution in a bank with an RB program executing in it. After the RB program finishes execution, a free area (the space once occupied by the RB program) remains in that bank of memory until the regular program completes execution.

3. It is totally occupied (completely used).

Prior to the execution of an RB program, the operating system searches for a partially used (2a or 2b) bank of memory and then, if none is available, for a completely empty one in which to execute the RB program.

Prior to the execution of a non-RB program, the operating system searches for a completely empty bank of memory and, if none is available, then for a partially used (2a only) one in which to execute the non-RB program.

In either case, the program to be run must fit in the available space.

Arguments and the top of user memory (location 6) are utilized by an RB program in the same manner as they are by non-RB programs. The operating system automatically allocates 256 bytes of stack area to each program. Locations 6 and 7 point to the top of these 256 bytes and, on entry to the program, the operating system loads the location into the stack pointer.

CROMIX SYSTEM CALL ERRORS

If the Cromix Operating System cannot complete a system call in the normal manner, for example, when a program tries to open a file which does not exist, an error condition is generated. This error condition is reflected by the state of the carry flag which is set or reset by the operating system when returning from a system call. If the carry flag is reset (=0), the system call completed its task successfully. If the carry flag is set (=1), the system call ended abnormally and the error type is returned in the **a** register. The **a** register may then be compared with a value from the error definition table in the **jsysequ.z80** file for user exception processing. The carry flag should be checked after every system call except for the **.exit** and **.error** calls. The **.exit** call does not return, and if the **.error** call returns an error, it is possible to generate an endless loop - an error routine which generates an error and then jumps to itself again.

If the **.error** system call is executed after a system call that generated an error, (carry set), an ASCII message equivalent to the error type is sent to the channel specified by the **b** register. (See the **.error** system call.)

The following example attempts to open a file that does not exist. When the file is not found, the program jumps to a create routine. Any other errors fall through to the **.error** system call, which displays the error on the console and then exits to the operating system.

```
        ld    hl,path_name      ; path name of file
        ld    c,OP.RDWR         ; read write access
        ld    d,0               ; non exclusive

        jsys .open              ; open the file

        jr    nc,open_ok        ; No errors, go to open_ok

        cp    ?notexist         ; if file not found
        jr    z,create_it       ;   go to create routine
                                ; else let system process the error
        ld    b,stderr          ;   stderr channel for console
        jsys .error             ;   send the error to the console
        jsys .exit              ;   exit to operating system

open_ok: jsys .exit             ; dummy open routine,
                                ; exit to operating system

create_it: jsys .exit           ; dummy create routine,
                                ; exit to operating system

path_name: defb '/usr/file_which_does_not_exist',0
```

Error Conditions

If the Cromix Operating System cannot complete a system call in the normal manner, an error is generated. The operating system flags an error condition by setting the carry bit in the flag register (the carry flag). A normal return from a system call is indicated by a reset carry flag.

If an error has occurred (carry flag is set or is equal to one), the a register contains the error code. The type of error that was returned may be established by comparing the **a** register with the following list of error codes. Each error code is preceded by the error number.

29 **?arglist**	The argument list that was provided is incorrect.
28 **?argtable**	The argument table is exhausted.
15 **?badcall**	The system call that was specified is illegal.
1 **?badchan**	An invalid channel number was specified. The operating system must be called with a channel number assigned at the time a file was opened or created.
42 **?badfree**	A block is out of range in the free list.
43 **?badinum**	The inode number is out of range.
8 **?badname**	The filename that was specified does not conform to proper filename syntax. The name is too long or contains illegal characters.
47 **?badpipe**	An attempt was made to write to a broken pipe.
34 **?badvalue**	The specified value was out of range.
40 **?chnaccess**	An attempt has been made to access a channel which the current user may not access.
36 **?devopen**	A device open error has occurred.
49 **?deadlock**	A possible deadlock condition has been detected.

31	**?difdev**	There is a cross device link. File references cannot exist across disks.
9	**?diraccess**	An attempt has been made to access a directory which the current user may not access. Make sure the pathname does not include any directories with privileged access.
37	**?diruse**	An attempt was made to delete a directory that was in use. All files must be deleted from a directory before it may be deleted.
4	**?endfile**	An end of file condition exists on the file being processed. There is no data in the file beyond (in a forward direction from) the current file position.
11	**?exists**	An attempt has been made to create a file that already exists.
10	**?filaccess**	An attempt has been made to open a file to which the current user has no access.
16	**?filsize**	The size of the file is too big.
6	**?filtable**	The file table has been exhausted.
38	**?filuse**	The requested file is an exclusive access file and was in use.
22	**?fsbusy**	The requested file system was busy.
14	**?inotable**	The inode table is exhausted.
5	**?ioerror**	A physical data transmission error has occurred.
19	**?isdir**	The specified pathname is that of a directory.
50	**?lcktable**	The lock table is exhausted.
49	**?locked**	The specified sequence is already locked.
17	**?mnttable**	The mount table is exhausted.
32	**?nodevice**	There is no device driver for the referenced device.
25	**?nochild**	There is no child process.

13	**?noinode**	No inodes are left.
39	**?nomatch**	There is no match on the specified ambiguous pathname.
26	**?nomemory**	There is not enough memory.
45	**?noproc**	The process does not exist.
12	**?nospace**	An attempt has been made to write to a full disk.
21	**?notblk**	The specified device is not a block special device.
35	**?notconn**	The requested I/O device was not connected to the system.
41	**?notcromix**	The specified disk is not compatible with the Cromix Operating System.
18	**?notdir**	The specified pathname was not that of a directory.
7	**?notexist**	The specified file does not exist. Make sure that the pathname properly identifies the desired file.
24	**?notmount**	The specified device was not mounted prior to the call.
3	**?notopen**	The specified channel has not been opened or was closed prior to the system call. A file must be opened (using the **.open** or **.create** call) prior to being used for I/O.
23	**?notordin**	The requested file is not an ordinary file.
30	**?numlinks**	This operation would have created too many links to the specified file or device.
27	**?ovflo**	An overflow occurred during a divide operation.
20	**?priv**	An attempt was made to invoke a privileged system call by other than a privileged user.
44	**?readonly**	The device is mounted for read access only.

46	**?signal**	The system call was aborted.
2	**?toomany**	All possible channels are already open.
33	**?usrtable**	The user process table is exhausted.

system call: **.ALARM**
number: 43h
purpose: This call sends an alarm signal to the calling process.

user access: all users

summary:
```
hl    =   number of seconds
jsys      .alarm
```

calling parameters: hl The hl register pair contains either the number of seconds before a signal is sent to the current process or a zero to cancel a previous alarm.

return parameters: none

possible errors: none

The **.alarm** call sends an alarm signal to the current process after the specified number of seconds has elapsed. If the hl register pair is loaded with 0 (hl=0) and the **.alarm** call is executed after an alarm has been set, the previous alarm is canceled.

system call:	**.CACCESS**
number:	27h
purpose:	This call tests channel access.
user access:	all users
summary:	b = channel c = access bits jsys .caccess
calling parameters:	b The b register contains the number of the channel whose access is to be tested. c The c register contains the access bits to be tested. These bits can be ANDed together to test for various combinations of access privileges. These bits may be represented by: ^AC.READ read ^AC.EXEC execute ^AC.WRIT write ^AC.APND append
return parameters:	The carry flag is reset (=0) if the channel is open for the specified access. The carry flag is set (=1) and the a register contains the error code ?filaccess if the channel is not open for the specified access.
possible errors:	?filaccess ?notopen

The **.caccess** call tests the access privileges of an open channel.

Example:

CHANNEL ACCESS system call (jsys .caccess)

```
;
;       It is assumed that a channel was previously opened
;       and the channel number is in the B register.
;       (see OPEN system call)
;
; Request the channel access for the channel in the B register.
;

                ld      c,^AC.READ      ; test for read access

                jsys    .caccess        ; test channel access system call

; Registers returned:
;       C flag - set if specified access is not permitted
;       C flag - reset if specified access is permitted




;       ************    end of example  ************
```

system call:	**.CCHSTAT**
number:	23h
purpose:	This call changes the status of an open file.
user access:	see table

summary:

```
b     =   channel
c     =   status type (see table)
de    =   new value
jsys     .cchstat
```

calling parameters:

b — The b register contains the channel number associated with the open file.

c — The c register contains the status type to be changed.

For access privilege changes:

d — The d register contains the new value of the specified status type.

e — The e register contains a mask of the status bits to be changed.

^AC.READ	read
^AC.EXEC	execute
^AC.WRIT	write
^AC.APND	append

For other status changes:

de — The de register pair contains the new value.

return parameters: none

possible errors:

?filaccess
?priv
?notopen

The **.cchstat** call changes the access privileges associated with a file, the times associated with a file, the owner id of a file, or the group id of a file. Please refer to the following table.

The file must be open; the channel number is used to identify the file.

Table of Cchstat Calls

Who*	C Register	Status Type	Location of New Information
p	ST.OWNER	owner id	de = new value
p	ST.GROUP	group id	de = new value
p&o	ST.AOWNER	access owner	d = new value, e = mask
p&o	ST.AGROUP	access group	d = new value, e = mask
p&o	ST.AOTHER	access public	d = new value, e = mask
p	ST.TCREATE	time created	de -> 6 byte buffer
p	ST.TMODIFY	time last modified	de -> 6 byte buffer
p	ST.TACCESS	time last accessed	de -> 6 byte buffer
p	ST.TDUMPED	time last dumped	de -> 6 byte buffer

*p = privileged user
o = owner

Example:

CHANGE CHANNEL STATUS system call (jsys .cchstat)

```
;
;       It is assumed that a channel was previously opened
;       and the channel number is in the B register.
;       (see OPEN system call)
;
; Change the current owner to owner 5.
;
                ld      c,ST.OWNER      ; change owner to
                ld      de,5            ; owner number 5

                jsys    .cchstat        ; change channel status system call

; Registers returned:
;       none


;       ************    end of example  ************
```

system call:	**.CHDUP**
number:	0Ah
purpose:	This call duplicates a channel.
user access:	all users
summary:	b = existing channel jsys .chdup c = duplicate channel
calling parameters:	b The b register contains the existing channel number.
return parameters:	c The c register contains the duplicate channel number assigned by the system.
possible errors:	?notopen

The **.chdup** call duplicates a channel and may be used for channel number manipulation. Please refer to the **.pipe** system call for additional information.

system call:	**.CHKDEV**
number:	07h
purpose:	This call verifies the presence of a specified device driver in the operating system.
user access:	all users

summary:

```
c     =   type of device (block/char)
d     =   major device number
e     =   minor device number
jsys     .chkdev
```

calling parameters:

c — The c register indicates the type of device:

IS.BLOCK	block device
IS.CHAR	character device

d — The d register contains the major device number.

e — The e register contains the minor device number.

return parameters: none

possible errors: ?nodevice

The **.chkdev** call verifies the presence of a device driver. If the device driver is present in the operating system, the **.chkdev** call returns without an error (the carry flag is reset (=0)). If the device driver is not present, the carry flag is set (=1) by the call (an error is returned).

system call: **.CLINK**
number: 25h
purpose: This call establishes an additional link to an open file.

user access: all users

summary:
```
b    =   channel
de   ->  new pathname
jsys     .clink
```

calling parameters:

b — The b register contains the channel number of the open file.

de — The de register pair points to the file pathname to be established (the new pathname). The pathname must be terminated by a null character.

return parameters: none

possible errors:
?badname
?isdir
?numlinks
?diraccess

The **.clink** call establishes a link from the file open on the specified channel to the new file pathname. The new file pathname must not exist before the **.clink** call is made.

Example:

CHANNEL LINK system call (jsys .clink)

```
;
;       It is assumed that a channel was previously opened
;       and the channel number is in the B register.
;       (see OPEN system call)
;
; Make a link from the new pathname specified by the DE register pair
; to the file specified by the B register.
;

                ld      de,path_name    ; pointer to new name

                jsys    .clink          ; channel link system call

; Registers returned:
;       none

path_name:      defb    '/usr/new_path_name',0

;       ************    end of example  ************
```

system call:	**.CLOSE**
number:	0Bh
purpose:	This call closes an open file.
user access:	all users
summary:	b = channel jsys .close
calling parameters:	b The b register contains the channel number of the open file.
return parameters:	none
possible errors:	?notopen

The **.close** call flushes all buffers associated with the specified channel number and disassociates the channel number from the file to which it was assigned.

system call: **.CREATE**
number: 08h
purpose: This call creates and opens a file.

user access: all users

summary:

```
hl    ->   pathname
c     =    access mode
d     =    exclusive mode
jsys      .create
b     =    channel
```

calling parameters:

hl — The hl register pair points to a buffer containing the pathname of the file to be created and opened. The pathname must be terminated by a null character.

c — The c register contains the access mode value for opening the file. The following labels represent the values of the c register required to establish the desired access mode. The specified access mode is applicable to the current process.

Nonexclusive access values:

OP.READ	read only
OP.WRITE	write only
OP.RDWR	read/write
OP.APPEND	append

Exclusive access values:

OP.XREAD	read only
OP.XWRITE	write only
OP.XRDWR	read/write
OP.XAPPEND	append

If exclusive access is desired, one of the four exclusive access values listed above must be loaded into the c register. This, in conjunction with the desired exclusion bit(s) in the d register, denies other users access.

The following values may be ORed with the desired access value (see above) to select the truncate or

conditional options.

Truncate flag:

OP.TRUNCF	delete existing data

Conditional flag

OP.CONDF	return error if file exists

d The d register contains the mask for exclusive access. It is inspected only if the c register indicates exclusive access. Each of the specified bits must be set to prevent the file from being opened by another process for the specified access. (For example, ^OP.READ indicates that no other process may open the file with read access. This does not exclude another process from opening the file for read/write access. To exclude all reads, ^OP.READ and ^OP.RDWR must be ORed together.) The following bits may be ORed together to set more than one bit.

Exclusive access bits:

^OP.READ	exclude read
^OP.WRITE	exclude write
^OP.RDWR	exclude read/write
^OP.APPEND	exclude append

return parameters: b The b register contains the channel number that the system assigned to the file.

possible errors:

?filtable
?badname
?diraccess
?isdir

The **.create** call creates a file with the specified pathname.

If the file does not exist at the time of the system call, it is created and opened with the requested access.

If the file does exist and the conditional flag is set, an error is returned. If the file does exist and the conditional flag is reset, the file is opened.

If the file exists and is opened (as specified by the conditional flag), the existing data is kept if the truncate flag is reset. The data is discarded (the file is truncated) if the truncate flag is set. A file may only be truncated if the user has write access to the file.

The channel number that the Cromix Operating System returns is used for subsequent access to the file.

The file created has default access privileges. In a standard system, these are read and execute for group and public, and read, execute, write, and append for the owner.

Example:

CREATE FILE system call (jsys .create)

```
; The operating system  creates, opens, and assigns a channel number
; to the file /usr/mylib/test. The channel number
; is returned in the B register.
;
; File access:  read only
;               non exclusive
;

                ld      hl,path_name    ; pointer to pathname
                ld      c,OP.READ       ; access mode = read only
                ld      d,0             ; non exclusive

                jsys    .create         ; create file system call

; Registers returned:
;       B = Channel


path_name:      defb    '/usr/mylib/test',0


;       ************    end of example  ************
```

Example:

CREATE FILE system call (jsys .create)

```
; The operating system  creates, opens, and assigns a channel number
; to the file /usr/mylib/test. The channel number
; is returned in the B register.
;
; File access:  read only
;               exclude all other read access
;

                ld      hl,path_name            ; pointer to pathname
                ld      c,OP.XREAD              ; access mode = read only
                ld      d,^OP.READ|^OP.RDWR     ; exclude other users
                                                ; from reading the file

                jsys    .create                 ; create file system call

; Registers returned:
;       B = Channel

path_name:      defb    '/usr/mylib/test',0

;       ************    end of example  ************
```

Example:

CREATE FILE system call (jsys .create)

```
; The operating system  creates, opens and assigns a channel number
; to the file /usr/mylib/test. The channel number
; is returned in the B register.
;
; File access:  read/write
;               exclude all other read, write, and append access
;
;

                ld      hl,path_name                ; pointer to pathname
                ld      c,OP.XRDWR                  ; access mode = read/write
                ld      d,^OP.READ | ^OP.RDWR | ^OP.WRITE | ^OP.APPEND
                                                    ; exclude other users
                                                    ; from reading, writing,
                                                    ; or appending the file

                jsys    .create                     ; create file system call

; Registers returned:
;       B = Channel


path_name:      defb    '/usr/mylib/test',0


;       ************    end of example   ************
```

system call:	**.CSTAT**
number:	21h
purpose:	This call returns the status of an open file.
user access:	all users

summary:

```
b     =   channel
c     =   desired information
de    =   buffer
jsys     .cstat
de    =   return value
hl    =   return value
```

calling parameters:

- **b** The b register contains the channel number associated with the open file.
- **c** The c register contains the request to the system for the desired information. Refer to the table.
- **de** The de register pair may point to a 6 or 128-byte buffer. Refer to the table.

return parameters:

- **dehl** The de (and in some cases the hl) register pair contains the requested information. Refer to the table.

possible errors: ?notopen

The **.cstat** call returns channel status information. The file must be open; the channel number is used to identify the file. Please refer to the following table of **.cstat** calls.

Table of Cstat Calls

C Register	Information Returned	Location of Information
ST.ALL	all of inode	de -> 128 byte inode buffer
ST.OWNER	owner id	de
ST.GROUP	group id	de
ST.AOWNER	access owner	d
ST.AGROUP	access group	d
ST.AOTHER	access public	d
ST.FTYPE	file type	d = IS.ORDIN IS.DIRECT IS.CHAR IS.BLOCK
ST.SIZE	file size	dehl
ST.NLINKS	number of links	de
ST.INUM	inode number	de
ST.TCREATE	time created	de -> 6 byte buffer
ST.TMODIFY	time last modified	de -> 6 byte buffer
ST.TACCESS	time last accessed	de -> 6 byte buffer
ST.TDUMPED	time last dumped	de -> 6 byte buffer
ST.DEVNO	device number	d = major device number e = minor device number
ST.DEVICE	device number	d = major device number e = minor device number

ST.DEVNO returns the device numbers of the device specified by a device file. If the specified file is not a device file, ST.DEVNO returns zeros. ST.DEVICE returns the device numbers of the device on which the specified file resides.

Example:

CHANNEL STATUS system call (jsys .cstat)

```
;
;       It is assumed that a channel was previously opened
;       and the channel number is in the B register.
;       (see OPEN system call)
;
; Request the file size for the channel specified by the B register.
;

                ld      c,ST.SIZE       ; request file size

                jsys    .cstat          ; test channel status system call

; Registers returned:
;       DEHL = file size


;       ************    end of example   ************
```

system call:	**.DELETE**
number:	06h
purpose:	This call deletes a directory entry.
user access:	all users
summary:	hl -> pathname jsys .delete
calling parameters:	hl The hl register pair points to a buffer containing the pathname of the directory or file to be deleted. The pathname must be terminated by a null character.
return parameters:	none
possible errors:	?diraccess ?notexist ?badname

The **.delete** call attempts to remove the specified directory entry. If the removed directory entry is the last link to the file, the file itself is deleted, the space occupied by the file is released, and its contents lost.

Write access is required to delete the directory entry.

If the file is open at the time the system call is made and the specified directory entry is the last link to the file, the directory entry is deleted immediately. The file itself is not deleted until the active process closes the file.

In order for a directory to be deleted, it must not

1. Contain any files;
2. Be the current directory for any user; or
3. Be the root directory of a device.

Example:

DELETE system call (jsys .delete)

```
;
; The operating system deletes
; the file specified by the HL register pair (/usr/mylib/test).
;
;

                ld      hl,path_name     ; pointer to pathname

                jsys    .delete          ; delete file system call

; Registers returned:
;       none

path_name:      defb    '/usr/mylib/test',0

;       ************    end of example  ************
```

system call:	**.DIVD**
number:	54h
purpose:	This call divides one unsigned integer by another.
user access:	all users

summary:

```
dehl =   dividend
bc   =   divisor
jsys    .divd
hl   =   quotient
de   =   remainder
```

calling parameters:

dehl The dehl registers contain an unsigned 32-bit integer. This is the dividend.

bc The bc register pair contains an unsigned 16-bit integer. This is the divisor.

return parameters:

hl The hl register pair returns the quotient.

de The de register pair returns the remainder.

possible errors: ?ovflo

The **.divd** call divides one unsigned integer by another and returns the quotient and remainder.

```
dividend (dehl)
---------------  =  quotient (hl) and remainder (de)
divisor  (bc)
```

Example:

DIVIDE system call (jsys .divd)

```
;
;
; Divide 2000 by 256
;
;    {2000/256}

                ld      de,0            ; dividend = 2000
                ld      hl,2000         ;  /
                ld      bc,256          ; divisor = 256

                jsys    .divd           ; divide system call

; Registers returned:
;       HL = quotient
;       DE = remainder
;



;       ************    end of example  ************
```

system call:	**.ERROR**
number:	1Ch
purpose:	This call displays an error message.
user access:	all users

summary:

```
a      =   the error number
b      =   channel
c          }
de         }as returned by the call
hl         }generating the error
jsys     .error
```

calling parameters:

a — The a register contains the error number generating the error.

b — The b register contains the channel number. This channel receives the error message and is usually set to **stderr**.

all — All registers except the b, prime, and index registers remain as returned by the system call that generated the error.

return parameters: none

possible errors:

The **.error** call sends an error message to the specified channel. It should only be called immediately after a system call that generated an error (if the carry bit in the flag register has been set).

Errors may occur during calls to error; this sets the carry bit. (Refer to the section of this chapter titled Cromix System Call Errors.)

system call:	**.EXCHG**
number:	0C
purpose:	This call exchanges filenames of two open files.
user access:	all users
summary:	b = channel number c = channel number jsys .exchg
calling parameters:	b&c The b and c registers contain channel numbers of two open files.
return parameters:	none
possible errors:	?notopen

The **.exchg** call exchanges the filenames of two open files. After **.exchg** is executed, the two filenames remain associated with their original inodes, but the block pointers of the inodes are changed.

system call:	**.EXEC**
number:	4Ch
purpose:	This call executes a program.
user access:	all users
summary:	de -> argument list hl -> pathname jsys .exec
calling parameters:	de The de register pair points to a list of pointers. The list of pointers is terminated by a null pointer. Each pointer points to a null terminated character string. Each string is an argument passed to the new program. hl The hl register pair points to the pathname of the file to be executed. A null character terminates the pathname.
return parameters:	none (does not return)
possible errors:	?notexist ?filaccess ?nomemory

The **.exec** call attempts to load the new program in a free memory area. If there is no memory available, the **?nomemory** error is returned.

Any channels opened before the execution of the **.exec** system call are passed to the new process.

Example:

EXECUTE system call (jsys .exec)

```
;
; The operating system executes the program specified by
; the HL register pair and passes the arguments specified by the
; DE register pair.
;
;

                ld      hl,path_name     ; pointer to program pathname
                ld      de,arg_list      ; pointer to argument list

                jsys    .exec            ; execute program system call

; Registers returned:
;       none (.exec does not return)


arg_list:       defw    argum0           ; pointer to argument zero
                defw    arguml           ; pointer to argument one
                defw    argum2           ; pointer to argument two
                defw    0                ; end of argument pointer list

argum0          defb    'mode',0         ; argument zero (used by PSTAT)
arguml          defb    'ttyl',0         ; argument one
argum2          defb    '-pa',0          ; argument two

path_name:      defb    '/bin/mode.bin',0     ; program name to execute

;       ************    end of example   ************
```

system call:	**.EXIT**
number:	46h
purpose:	This call exits from a process.
user access:	all users
summary:	hl = termination status jsys .exit
calling parameters:	hl The hl register pair contains the termination status to be passed back to the calling program. 0 termination OK 1 abnormal termination
return parameters:	none
possible errors:	

The **.exit** call provides an exit from an active process. It closes all channels and unlocks all locks that the current process initiated.

The Shell **If -err** construction tests the termination status of the last program executed.

system call:	**.FACCESS**
number:	26h
purpose:	This call tests file access.
user access:	all users
summary:	c = access bits hl -> pathname jsys .faccess
calling parameters:	c The c register contains the access bits to be tested. These bits can be ORed together to test for various combinations of access privileges. These bits may be represented by: ^AC.READ read ^AC.EXEC execute ^AC.WRIT write ^AC.APND append hl The hl register pair points to the pathname of the file to be tested. The pathname must be terminated by a null character.
return parameters:	The carry flag is set (=1) and the a register contains the error code ?filaccess if the file may not be accessed as specified. The carry flag is reset (=0) if the file may be accessed as specified.
possible errors:	?badname ?filaccess ?notexist

The **.faccess** call tests the access privileges of a file.

Example:

FILE ACCESS system call (jsys .faccess)

```
;
; Retrieve the file access status of the file specified by the HL register
; pair.
;
;
                ld      c,^AC.READ      ; test read access
                ld      hl,path_name    ; pointer to pathname

                jsys    .faccess        ; test file access system call

; Registers returned:
;       C flag - set if specified access is not permitted
        C flag - reset if specified access is permitted



path_name:      defb    '/usr/my_test_file',0

;       ************    end of example  ************
```

system call:	**.FCHSTAT**
number:	22h
purpose:	This call changes the status of a file.
user access:	see table

summary:

```
c     =   status type (see table)
de    =   new value
hl    ->   pathname
jsys    .fchstat
```

calling parameters:

c The c register contains the status type to be changed.

For access privilege changes:

d The d register contains the new value of the specified status type.

e The e register contains a mask of the status bits to be changed.

^AC.READ	read
^AC.EXEC	execute
^AC.WRIT	write
^AC.APND	append

For other status changes:

de The de register pair contains the new value.

hl The hl register pair points to the pathname of the file whose status is to be changed.

return parameters: none

possible errors:

?filaccess
?notexist
?badname

The **.fchstat** call changes the access privileges associated with a file, the times associated with a file, the owner id of a file, or the group id of a file. Please refer to the following table.

Table of Fchstat Calls

Who*	C Register	Status Type	Location of New Information
p	ST.OWNER	owner id	de = new value
p	ST.GROUP	group id	de = new value
p&o	ST.AOWNER	access owner	d = new value, e = mask
p&o	ST.AGROUP	access group	d = new value, e = mask
p&o	ST.AOTHER	access public	d = new value, e = mask
p	ST.TCREATE	time created	de -> 6 byte buffer
p	ST.TMODIFY	time last modified	de -> 6 byte buffer
p	ST.TACCESS	time last accessed	de -> 6 byte buffer
p	ST.TDUMPED	time last dumped	de -> 6 byte buffer

*p = privileged user
o = owner

Example:

CHANGE FILE STATUS system call (jsys .fchstat)

```
;
;
; Change the owner of the file specified by the HL register pair to 5.
;
;
;

                ld      hl,path_name    ; pointer to pathname
                ld      c,ST.OWNER      ; change owner to
                ld      de,5            ; owner number 5

                jsys    .fchstat        ; change file status system call

; Registers returned:
;       none


path_name       defb    '/usr/my_test_file',0

;       ************    end of example      ************
```

system call: **.FEXEC**
number: 4Bh
purpose: This call forks and executes a program.

user access: all users

summary:

```
b     =    new process signal mask
c     =    new process signals values
de    ->   argument list
hl    ->   pathname
jsys       .fexec
```

calling:
parameters:

b — The b register contains an 8-bit mask which indicates what signals to pass to the child (new) process. If a bit is reset (=0) then either: the child will ignore or be aborted by the signal corresponding to that bit, depending upon whether the parent ignores or is aborted by the signal; or the child will be aborted by the signal if the parent has provided a trapping routine (i.e., with the **.signal** call). If a bit is set (=1), the corresponding bit of the c register determines what the child process does with the corresponding signal.

c — If the corresponding bit in the b register is set (=1), the bit in the c register indicates the action to be taken by the child process when the corresponding signal is received. A bit that is reset (=0) causes the child process to abort when that signal is received. A bit that is set (=1) causes that signal to be ignored. The kill signal cannot be masked.

de — The de register pair points to a list of pointers. A null pointer terminates the list of pointers. Each pointer points to a null terminated character string. Each string is an argument passed to the new program.

	hl	The hl register pair points to the pathname of the file to execute. The pathname is terminated by a null character.
return parameters:	hl	The hl register pair contains the child process id (PID) number.
possible errors:	?notexist ?filaccess ?badname ?nomemory	

The **.fexec** call begins execution of a program and returns control to the calling program. This call is similar to the **.exec** call, except that a new process is created.

Notes

For related information pertaining to signals refer to the **Signals** section at the beginning of this chapter.

Example:

FORK AND EXECUTE system call (jsys .fexec)

```
;
; The operating system creates a new process,
; begins execution of the file specified by the HL register
; pair, and returns to the current process. The arguments
; specified by the DE register pair are passed to the
; new process.

                ld      b,0             ; same signals as parent
                ld      de,arg_list     ; pointer to argument list
                ld      hl,path_name    ; pointer to program pathname

                jsys    .fexec          ; fork and execute system call


; Registers returned:
;       HL = child process number



arg_list:       defw    argum0          ; pointer to argument zero
                defw    argum1          ; pointer to argument one
                defw    argum2          ; pointer to argument two
                defw    0               ; end of argument pointer list

argum0:         defb    'mode',0        ; argument zero (used by PSTAT)
argum1:         defb    'B',0           ; argument one
argum2:         defb    '9600',0        ; argument two

path_name:      defb    '/bin/mode.bin',0       ; name of file

;       ************    end of example  ************
```

system call:	**.FLINK**
number:	24h
purpose:	This call establishes a link to a file.
user access:	all users
summary:	de -> new pathname hl -> old pathname jsys .flink
calling parameters:	de The de register pair points to the new file pathname to be established. The pathname is terminated by a null character. hl The hl register pair points to the existing file pathname. The pathname is terminated by a null character.
return parameters:	none
possible errors:	?badname ?isdir ?numlinks ?diraccess ?notexist

The **.flink** call establishes a link to a file.

Example:

FILE LINK system call (jsys .flink)

```
;
;
; Create a new file pathname specified by the HL register pair
; and link to an existing file pathname specified by the DE register
; pair.
;
;

                ld      hl,old_path     ; pointer to old pathname
                ld      de,new_path     ; pointer to new pathname

                jsys    .flink          ; file link system call


; Registers returned:
;    none



old_path:       defb    '/usr/old_file_name',0
new_path:       defb    '/usr/new_file_name',0


;       ************    end of example  ************
```

system call:	**.FSHELL**
number:	48h
purpose:	This call forks a Shell process.
user access:	all users

summary:

```
b       =    new process signal mask
c       =    new process signals values
de      ->   argument list
jsys        .fshell
hl           PID number
```

calling parameters:

b — The b register contains an 8-bit mask which indicates what signals are to be passed to the child (new) process. If a bit is reset (=0) then either: the child will ignore or be aborted by the signal corresponding to that bit, depending upon whether the parent ignores or is aborted by the signal; or the child will be aborted by the signal if the parent has provided a trapping routine (i.e., with the **.signal** call). If a bit is set (=1), the corresponding bit of the c register determines what the child process does with the corresponding signal.

c — If the corresponding bit in the b register is set (=1), the bit in the c register indicates what the child process should do when the corresponding signal is received. A reset bit (=0) causes the child process to abort when the signal is received. A bit that is set (=1) causes that signal to be ignored. The kill signal, sigkill, cannot be masked.

de — The de register pair points to a list of pointers. The list of pointers is terminated by a null pointer. Each of the pointers points to a null terminated character string. Each string is an argument passed to the new program.

return parameters:	hl	The hl register pair contains the new process id (PID) number.

possible errors:	?notexist ?filaccess

The **.fshell** call initiates execution of a child Shell process which acquires a new PID.

Options

These options are needed only when a program is calling a Shell. They are not useful when a Shell is called from the terminal.

The **-c** option indicates that the command line being passed to the Shell has not been parsed.

The **-p** option indicates that the command line being passed to the Shell has been parsed.

The **-q** option requests that lines from a command file not be echoed to the terminal (standard output).

Notes

For related information pertaining to signals refer to the **Signals** section at the beginning of this chapter.

The **.fshell** call expects arguments to be in one of the following three forms:

<u>Form 1</u> (passing command filenames)

```
de ->   arg 0  ->  "shell \0"
        arg 1  ->  arg 1 (a command filename)
        arg 2  ->  arg 2
        .
        .
        .
        0
```

```
Form 2      (passing a parsed argument list)

  de ->     arg 0   ->   "shell \0"
            arg 1   ->   -p \0"
            arg 2   ->   command name (terminated by \0)
            arg 3   ->   command's first argument (terminated by \0)
            arg 4   ->   command's second argument (terminated by \0)
            .
            .
            .
            .
            0

Form 3      (passing a command line)

  de ->     arg 0   ->   "shell \0"
            arg 1   ->   "-c \0"
            arg 2   ->   command line (terminated by \0)
            0
```

Example:

FORK A SHELL system call (jsys .fshell)

```
;
; The operating system creates a new process,
; begins execution of the command file specified by the
; command argument, and returns to the current process.
; The specified arguments (arguments one & two) are
; passed to the new process.

                ld      b,0             ; same signals as parent
                ld      de,arg_list     ; pointer to argument list

                jsys    .fshell         ; fork a shell system call


; Registers returned.
;       HL = process number


arg_list:       defw    arg_0           ; pointer to argument 0
                defw    arg_1           ; pointer to argument 1
                defw    arg_2           ; pointer to argument 2
                defw    arg_3           ; pointer to argument 3
                defw    0               ; end of argument pointer list

arg_0           defb    'sh',0          ; request a shell
arg_1           defb    '-p',0          ; parse arguments
arg_2           defb    'mode',0        ; program name
arg_3           defb    'prt',0         ; request mode of device prt


;       ************       end of example     ************
```

system call:	**.FSTAT**	
number:	20h	
purpose:	This call returns the status of a file.	
user access:	all users	

summary:

```
c     =    desired information
de   ->    buffer
hl   ->    pathname
jsys      .fstat
de    =    return value
hl    =    return value
```

calling parameters:

- c — The c register contains the request for the desired system information. Refer to the table.
- de — The de register pair may point to a 6 or 128-byte buffer. Refer to the following table.
- hl — The hl register pair points to the pathname of the file whose status is to be checked.

return parameters:

- dehl — The de (and in some cases the hl) register pair contains the requested information. Refer to the table.

possible errors: ?badname

The **.fstat** call returns file status information. Please refer to the following table of **.fstat** calls.

Table of FSTAT Calls

C Register	Information Returned	Location of Information
ST.ALL	all of inode	de -> 128 byte inode buffer
ST.OWNER	owner id	de
ST.GROUP	group id	de
ST.AOWNER	access owner	d
ST.AGROUP	access group	d
ST.AOTHER	access public	d
ST.FTYPE	file type	d = IS.ORDIN IS.DIRECT IS.CHAR IS.BLOCK
ST.SIZE	file size	dehl
ST.NLINKS	number of links	de
ST.INUM	inode number	de
ST.TCREATE	time created	de -> 6 byte buffer
ST.TMODIFY	time last modified	de -> 6 byte buffer
ST.TACCESS	time last accessed	de -> 6 byte buffer
ST.TDUMPED	time last dumped	de -> 6 byte buffer
ST.DEVNO	device number	d = major device number e = minor device number
ST.DEVICE	device number	d = major device number e = minor device number

ST.DEVNO returns the device numbers of the device specified in a device file. If the specified file is not a device file, ST.DEVNO returns zeros. ST.DEVICE returns the device numbers of the device on which the specified file resides.

Example:

FILE STATUS system call (jsys .fstat)

```
;
;
; Request the file size of the file specified by the HL register pair.
;
;

                ld      hl,path_name    ; pointer to the pathname
                ld      c,ST.SIZE       ; request file size

                jsys    .fstat          ; test file status system call

; Registers returned:
;       DEHL = file size

path_name:      defb    '/usr/my_test_file',0


;       ************    end of example  ************
```

system call:	**.GETDATE**
number:	30h
purpose:	This call returns the date.
user access:	all users

```
summary:     jsys    .getdate
             d    =   day of the week
             e    =   year
             h    =   month
             l    =   day of the month
```

calling parameters: none

return parameters:

d — The d register contains the day of the week where 1 represents Sunday, 2 represents Monday, etc.

e — The e register contains the year minus 1900. This means 1980 is represented as 80 and 2004 is 104.

h — The h register contains the month where 1 represents January, 2 represents February, etc.

l — The l register contains the day of the month in the range between 1 and 31 inclusive.

possible errors:

The **.getdate** call returns the current date as recorded by the Cromix system clock.

system call:	**.GETDIR**
number:	02h
purpose:	This call returns the current directory pathname.
user access:	all users
summary:	hl -> buffer jsys .getdir
calling parameters:	hl The hl register pair points to a 128-byte buffer for the current directory pathname.
return parameters:	none
possible errors:	

The **.getdir** call returns the pathname of the current directory.

Example:

GET DIRECTORY system call (jsys .getdir)

```
;
;
; Request the current directory name from the operating system.
;
;
;
;
                ld      hl,buffer       ; pointer to 128 byte buffer

                jsys    .getdir         ; get directory system call

; Registers returned:
;       none



buffer:         defs    128             ; 128 byte directory buffer


;       ************    end of example  ************
```

system call:	**.GETGROUP**
number:	36h
purpose:	This call returns the group id.
user access:	all users

summary:

```
c      =   id type
jsys       .getgroup
hl     =   group id
```

calling parameters:	c	The c register contains a value indicating the type of identification desired. ID.EFFECTIVE ID.LOGIN ID.PROGRAM
return parameters:	hl	The hl register pair contains the type of group identification requested.
possible errors:		

The **.getgroup** call returns the group id.

Example:

GET GROUP system call (jsys .getgroup)

```
;
;
; Request the login id from the operating system.
;
;
;
;

                ld      c,ID.LOGIN      ; request login id of group

                jsys    .getgroup       ; get group system call

; Registers returned:
;       HL = group id


;       ************   end of example  ************
```

system call: **.GETMODE**
number: 12h
purpose: This call returns the characteristics of a character device.

user access: all users

summary:

```
b     =   channel
c     =   mode type
jsys     .getmode
d     =   return value
```

calling parameters:

b — The b register contains the channel number of the opened device.

c — The c register contains the MODE TYPE to be tested. The c register may be loaded with one of the following MODE TYPES:

Mode types for printers and terminals:

C Register	Significance
MD_ISPEED	input speed (baud rate)
MD_OSPEED	output speed (baud rate)
MD_MODE1	flags: tandem, lcase, echo, crdevice, raw, odd, even, xtab
MD_MODED	delays: nldelay, tabdelay, crdelay, ffdelay, bsdelay
MD_MODE2	flags: pause, notimmecho, noecnl, sigenable, abenable, xff, wrap, sigallc
MD_MODE3	flags: escretn, fnkeys, hupenab, sighupall, cbreak, binary, discard
MD_ERASE	auxiliary input erase character
MD_DELECHO	deletion echo character
MD_LKILL	input line kill character
MD_USIGNAL	SIGUSER signal key
MD_LENGTH	page length

MD_WIDTH	page width
MD_BMARGIN	bottom margin
MD_STATUS	input buffer status
MD_IFLUSH	flush input buffers
MD_FNKEYS	enable or disable function keys

return parameters: d — The d register contains the value of the mode type specified by the c register.

possible errors:

The **.getmode** call returns the characteristics of a character device. Refer to the **.setmode** system call and the Mode utility for more information.

MD_STATUS getmode call

If the c register contains MD_STATUS, then the d register is returned with the following bits set according to the options in effect (0 = disabled, 1 = enabled).

Bit in D	Significance
INOTEMPTY	At least one character is in the input buffer. The character is available immediately if CBREAK, RAW, or BINARY mode is set. Otherwise the character is part of a line which will be available after a line terminator has been entered.

Example:

GET MODE system call (jsys .getmode)

```
;
; Request the status of the expand tab option of the current console.
;
;
;
;

                ld      b,stdout        ; current console channel number
                ld      c,MD_MODE1      ; mode type

                jsys    .getmode        ; setmode system call

; Registers returned:
;       D = requested mode status

; The desired MODE1 option bits may now be tested

                bit     xtab,d          ; expand tab option on?
                jr      nz,yes          ;       yes then continue
                                        ; else
                 .                      ;       process
                 .                      ;
                                        ;
yes:                                    ;


;       ************    end of example  ************
```

system call:	**.GETPOS**
number:	10h
purpose:	This call returns a file pointer.
user access:	all users
summary:	b = channel number jsys .getpos dehl = file pointer
calling parameters:	b The b register contains the channel number of the open file.
return Parameters:	dehl The de and hl register pairs contain the current value of the file pointer. This is a 32 bit unsigned integer.
possible errors:	?notopen

The **.getpos** call returns the logical position (byte value) of the file pointer of an open file.

system call:	**.GETPRIOR**
number:	38h
purpose:	This call returns the priority of the calling process.
user access:	all users
summary:	jsys .getprior l = priority number
calling parameters:	none
return parameters:	l The l register contains the priority number of the current process (-40 to 40).
possible errors:	

The **.getprior** call returns the priority number of the calling process. This number is in the range -40 (highest priority) to 40.

system call: **.GETPROC**
number: 3Ah
purpose: This call returns the PID of the calling process.

user access: all users

summary:

```
jsys    .getproc
hl   =   process id
```

calling parameters: none

return parameters: hl The hl register pair contains the process id.

possible errors:

The **.getproc** call returns the process id of the calling process.

```
    system call:     .GETTIME
         number:     32h
        purpose:     This call returns the time.

    user access:     all users

        summary:     jsys    .gettime
                     e    =   hour
                     h    =   minute
                     l    =   second


        calling
     parameters:     none


         return
     parameters:     e    The e register contains the hours
                          portion of the current time based on
                          a 24-hour clock (e.g., 6 p.m. is
                          represented as 18 hours).

                     h    The h register contains the minutes
                          portion of the current time.  This
                          is the number of minutes since the
                          current hour started.

                     l    The l register contains the seconds
                          portion of the current time.  This
                          is the number of seconds since the
                          current minute started.


       possible
         errors:
```

The **.gettime** call returns the current time as recorded by the Cromix system clock.

```
system call:    .GETUSER
     number:    34h
    purpose:    This call returns the user id of the
                calling process.

user access:    all users

    summary:    c     =   idtype
                jsys     .getuser
                hl    =   user id


    calling
 parameters:    c     The c register contains a value
                      indicating the type of
                      identification desired:

                      ID.EFFECTIVE
                      ID.LOGIN
                      ID.PROGRAM


     return
 parameters:    hl    The hl register pair contains the
                      requested user identification.


   possible
     errors:    none
```

The **.getuser** call returns the user id as specified.

Example:

GET USER system call (jsys .getuser)

```
;
;
; Request the current user login id from the operating system.
;
;
;
;
                ld      c,ID.LOGIN      ; request login id of user

                jsys    .getuser        ; get user system call

; Registers returned:
;       HL = user id


;       ************    end of example  ************
```

system call:	**.INDIRECT**
number:	51h
purpose:	This call executes the system call in a register.
user access:	all users

summary:

```
a     =   call number
bc    =   according to system call
de    =   according to system call
hl    =   according to system call
jsys      .indirect
```

calling parameters: a — The a register contains the system call number.

return parameters: none

possible errors: according to system call

The **.indirect** call executes the system call in a register.

Example:

INDIRECT system call (jsys .indirect)

```
;
;
; Execute the system call specified by the A registeı
;
;
;
;
;

                ld      a,.getuser      ; get user system call
                ld      hl,ID.LOGIN     ; request user login id

                jsys    .indirect       ; indirect system call

; Registers returned:
;       HL = user id




;       ************    end of example  ************
```

system call:	**.KILL**	
number:	41h	
purpose:	This call sends a signal to a process.	
user access:	all users	processes initiated by the user
	privileged user	any process

summary:

```
c      =   signal type
hl     =   process id
jsys      .kill
```

calling parameters:

c — The c register contains the type of signal that is sent.

SIGABORT	CNTRL-C signal
SIGUSER	user special key signal
SIGKILL	kill signal
SIGTERM	terminate signal
SIGALARM	alarm signal
SIGPIPE	broken pipe signal
SIGHANGUP	phone hangup signal

hl — The hl register pair contains the process id of the process to which the signal is sent.

return parameters: none

possible errors:

?priv
?noproc
?badcall

The **.kill** call sends a signal to a process. When any signal is received by a process, the process is aborted unless the **.signal** system call specifies that a subroutine be executed or that the signal be ignored.

When a signal is received, unless it is ignored, an unsatisfied request for input or output from a character device is canceled. Examples are reading a buffered line from a console or writing a line to a printer.

If a signal is sent to process 0, the same type of signal is sent to all processes that belong to the user invoking the call.

If the user is a privileged user and a SIGUSER signal is sent to process 1, system shutdown is initiated.

If a SIGABORT signal is sent to process 1, the **/etc/ttys** file is reexamined. If an entry has a **0** in the leftmost column, it is logged off and all of its processes are terminated. If an entry shows a **1** in that column, the terminal is logged in if it is not already logged in.

system call: **.LOCK**
number: 3Eh
purpose: This call assists in implementing record level file locks.

user access: all users

summary:

```
c       =    lock type
de      =    lock length
hl      ->   lock sequence
jsys       .lock
```

calling parameters:

c — The c register contains the type of lock to be implemented.

bit 0 — If bit 0 of the c register contains 0, the lock may not be shared; a 1 indicates that the lock may be shared. A shared lock may be used by more than one process.

bit 1 — If bit 1 of the c register contains 0, then the lock is unconditional; a 1 indicates that the lock is conditional. If a conditional lock fails, a ?locked error is returned. If an unconditional lock fails, the process is put to sleep until the lock does not fail. The word **fail** means a lock sequence matches the lock sequence of a prior lock still in effect in one of the following ways:

1. A nonshareable lock was requested when matching lock already existed.

2. A shareable lock was requested when a nonshareable matching lock already existed.

   A lock also fails if the lock table is full. This returns a ?lcktable error to the process. There is space for 16 locks.

de — The de register pair contains the length of the locking sequence. This must be a number between 1 and 16.

hl The hl register pair points to the locking sequence of 16 or fewer bytes.

return parameters: none

possible errors: ?locked
?deadlock
?lcktable

The **.lock** call helps implement record level file locks. This call allows the operating system to absorb some of the overhead involved in this procedure. No actual locks are imposed on files by the operating system. On a passive system such as this one, the application program must enforce the rule that all file access requires the lock mechanism.

If more than one program is relying on the **.lock** system call, a mutually agreed upon scheme must be devised so that all programs can reference records within a file by the same identifier. This identifier is the locking sequence and may be comprised of 1 to 16 bytes.

An example of a viable locking sequence is the first 8 bytes of the filename followed by the number of the record to be locked. This scheme works as long as no two files in use have names beginning with the same 8 characters, and as long as no two processes are using the same file through two links with different names.

A more elaborate locking scheme involves the use of file device and inode numbers. The combination of the device and inode numbers forms a unique file identifier that is the same no matter what link or name is used to refer to a file. Both the inode number and the device number on which the file resides can be obtained through the **.cstat** system call. The locking sequence is composed of a device number followed by an inode number and a record number.

After a certain lock sequence is locked, any other process attempting to lock the same sequence either receives an error message or is put to sleep until the sequence can be locked.

If the number of available locks is exceeded, the operating system returns from a **.lock** system call with a ?lcktable error. This error does not indicate anything about the lock sequence but only that there is no room left in the lock table.

The ?deadlock error is returned if the operating system detects a deadlock condition.

All locks installed by a process are automatically unlocked when the process is terminated.

To summarize, if a record must be locked so that no other process may use it, bit 0 must be set to 0, for a nonshareable lock. On reads where an update (writing back to the file) is required, use a nonshareable lock. If any shareable locks are in effect when the nonshareable lock is attempted, it fails. While a nonshareable lock is in effect, all other locks, whether shareable or nonshareable, fail.

A shareable lock is used in a case where a record may be read by anyone. For those reads not requiring an update, a shareable lock is appropriate.

When using an unconditional lock, the **.signal** system call must be used so that the process requesting the lock can be awaken by a SIGALARM signal from the operating system when the lock can be granted.

```
system call:     .MAKDEV
     number:     00h
    purpose:     This call creates a new name for a
                 device.

user access:     privileged user

    summary:     c     =    type of device (block/char)
                 d     =    major device number
                 e     =    minor device number
                 hl    ->   pathname
                 jsys     .makdev

    calling
 parameters:     c     The c register indicates the type of
                       device:

                       IS.BLOCK     block device
                       IS.CHAR      character device

                 d     The d register contains the major
                       device number.

                 e     The e register contains the minor
                       device number.

                 hl    The hl register pair points to the
                       new pathname for the device.  The
                       pathname must be terminated by a
                       null character.

     return
 parameters:     none

   possible
     errors:     ?badname
                 ?exists
```

The **.makdev** call assigns a label to an existing device in the operating system.

system call:	**.MAKDIR**
number:	01h
purpose:	This call creates a new directory.
user access:	all users
summary:	hl -> pathname jsys .makdir
calling parameters:	hl The hl register pair points to the pathname of the new directory. The pathname must be terminated by a null character.
return parameters:	none
possible errors:	?badname ?exists

The **.makdir** call creates a new directory.

Example:

MAKE DIRECTORY system call (jsys .makdir)

```
;
;
; Create the directory specified by the HL register pair.
;
;
;
;
                ld      hl,path_name       ; pointer to pathname

                jsys    .makdir            ; make directory system call

; Registers returned:
;       none


path_name:      defb    '/usr/my_new_directory',0       ; new directory name


;       ************    end of example  ************
```

system call: **.MOUNT**
number: 04h
purpose: This call enables access to a file system.

user access: privileged user

summary:

```
c     =   type of access
de    ->  block device pathname
hl    ->  dummy pathname
jsys     .mount
```

calling parameters:

c — The c register indicates the desired access:

0 read/write
1 read only

de — The de register pair points to a buffer containing the pathname of the block device on which the file system is to be mounted. The pathname must be terminated by a null character.

hl — The hl register pair points to a buffer containing the pathname of the dummy file in which the file system is to be mounted. The pathname must be terminated by a null character.

return parameters: none

possible errors:

?mttable
?fsbusy
?notblk
?badname
?notexist

The **.mount** call declares that a file system is to be mounted on a specified device. References to the file system pathname refer to the root file of the mounted file system.

The dummy file pathname is the file system pathname while the file system remains mounted. When the system is unmounted, the name reverts to dummy status.

Example:

MOUNT FILE SYSTEM system call (jsys .mount)

```
;
;
; Mount the device specified by the DE register pair
; to the file specified by the HL register pair.
;
;
;
;
;
                ld      hl,path_name    ; pointer to pathname
                ld      de,path_device  ; pointer to device
                ld      c,0             ; read/write access

                jsys    .mount          ; mount file system system call

; Registers returned:
;       none


path_name:      defb    '/a',0          ; specify the 'a' entry in the root

path_device:    defb    '/dev/fda',0    ; large floppy drive A


;       ************    end of example  ************
```

system call:	**.MULT**
number:	53h
purpose:	This call multiplies one integer by another.
user access:	all users
summary:	bc = multiplier hl = multiplicand jsys .mult dehl = product
calling parameters:	bc The bc register pair contains the multiplier. hl The hl register pair contains the multiplicand.
return parameters:	dehl The dehl registers contain an unsigned 32 bit integer. This is the product.
possible errors:	?ovflo

The **.mult** call multiplies one integer by another and returns the product.

Example:

MULTIPLY system call (jsys .mult)

```
;
;
; Multiply the number 256 by 2000
;
;   {256*2000}

                ld      bc,256          ; multiplier = 256
                ld      hl,2000         ; multiplicand = 2000

                jsys    .mult           ; multiply system call

; Registers returned:
;       DEHL = product
;



;       ************    end of example  ************
```

system call: **.OPEN**
number: 09h
purpose: This call opens a file for access.

user access: all users

summary:

```
c     =   access mode
d     =   exclusive mode
hl    ->  pathname
jsys     .open
b     =   channel
```

calling parameters:

c — The c register contains the access mode for opening the file.

Nonexclusive access values:

OP.READ	read only
OP.WRITE	write only
OP.RDWR	read/write
OP.APPEND	append

Exclusive access values:

OP.XREAD	read only
OP.XWRITE	write only
OP.XRDWR	read/write
OP.XAPPEND	append

If exclusive access is desired, one of the four exclusive access values listed above must be loaded into the c register. This, in conjunction with the desired exclusion bit(s) in the d register, excludes other user access.

d — The d register contains the mask for exclusive access. Each of the specified bits must be set to prevent the file from being opened by another process for the specified access. (For example, ^OP.READ indicates that no other process may open the file with read access. This does not exclude another process from opening the file for

read/write access. To exclude all reads, ^OP.READ and ^OP.RDWR must be ORed together.) The following bits may be ORed together to set more than one bit.

Exclusive access bits:

^OP.READ	exclude read
^OP.WRITE	exclude write
^OP.RDWR	exclude read/write
^OP.APPEND	exclude append

hl   The hl register pair points to a buffer containing the pathname of the file to be opened. The pathname must be terminated by a null character.

return parameters: b   The b register contains the channel number that the system has assigned to the file.

possible errors:

?filtable
?badname
?diraccess
?isdir

The **.open** call assigns a channel number to the specified file. The user is then allowed to read from and/or write to the file.

Example:

OPEN FILE system call (jsys .open)

```
;
; The operating system opens and assigns a channel
; to the file specified by the HL register pair (/usr/mylib/test).
; The channel number is returned in the B register.
;
; File access:  read only
;               non exclusive
;

                ld      hl,path_name     ; pointer to pathname
                ld      c,OP.READ        ; access mode = read only
                ld      d,0              ; non exclusive

                jsys    .open            ; open file system call

; Registers returned:
;       B = Channel

path_name:      defb    '/usr/mylib/test',0

;       ************    end of example  ************
```

Example:

OPEN FILE system call (jsys .open)

```
;
; The operating system opens and assigns a channel
; to the file specified by the HL register pair (/usr/mylib/test).
; The channel number is returned in the B register.
;
; File access:  read only
;               exclude all other read access
;

                ld      hl,path_name            ; pointer to pathname
                ld      c,OP.XREAD              ; access mode = read only
                ld      d,^OP.READ|^OP.RDWR     ; exclude other users
                                                ; from reading the file

                jsys    .open                   ; open file system call

; Registers returned:
;       B = Channel

path_name:      defb    '/usr/mylib/test',0

;       ************    end of example  ************
```

Example:

OPEN FILE system call (jsys .open)

```
;
; The operating system opens and assigns a channel
; to the file specified by the HL register pair (/usr/mylib/test).
; The channel number is returned in the B register.
;
; File access:  read/write
;               exclude all other read, write, and append access
;

                ld      hl,path_name            ; pointer to pathname
                ld      c,OP.XRDWR              ; access mode = read/write
                ld      d,^OP.READ | ^OP.RDWR | ^OP.WRITE | ^OP.APPEND
                                                ; exclude other users
                                                ; from reading, writing, or
                                                ; appending the file

                jsys    .open                   ; open file system call

; Registers returned:
;       B = Channel

path_name:      defb    '/usr/mylib/test',0

;       ************    end of example  ************
```

system call:	**.PAUSE**
number:	44h
purpose:	This call suspends execution and waits for a signal.
user access:	all users
summary:	jsys .pause
calling parameters:	none
return parameters:	none
possible errors:	none

The **.pause** call suspends execution of the current process until a signal, generated by a **.kill** or **.alarm** system call is received.

system call:	**.PIPE**
number:	0Eh
purpose:	This call creates a pipe.
user access:	all users
summary:	jsys .pipe b = input channel c = output channel
calling parameters:	none
return parameters:	b The b register contains the number of the input channel from which data is read out of the pipe. c The c register contains the number of the output channel to which data is written into the pipe.
possible errors:	?notopen

The **.pipe** call creates and opens an interprocess communication path called a pipe. Each pipe has two ends. One is the pipe's receiving end and the other is the pipe's sending end. The pipe's receiving end is the **output channel** returned from the **.pipe** system call, since it receives the output from a process. Similarly, the pipe's sending end is the **input channel** returned from the **.pipe** system call, since it sends input to a process. A pipe can be used for one-way communication between processes. Two pipes are required for bidirectional communication, one for each direction of communication.

Pipes are most frequently used for parent/child process communication. The balance of this description describes how this communication is typically set up.

When one process (the parent) initiates another (child) process, the child process inherits the standard input, output, and error channels of the parent. It is through manipulation of these channels, combined with the use of pipes, that parent/child communication is established.

The following procedure and sample program shows how a single pipe can be established so that the parent process can send data to the child. If it is necessary for the child to send data to the parent, another pipe will have to be opened in the same manner.

Three aspects of the Cromix Operating System make it possible to set up a pipe.

1. When a process is forked, the process inherits the standard input (channel 0), standard output (channel 1), and the standard error (channel 2) of the parent process.

2. When the Cromix system allocates a channel, it allocates the lowest channel number available.

3. A facility in the Cromix system allows a duplicate channel to be made of an open file.

The basic procedure for setting up a pipe from a process is outlined below, followed by a step by step description of what occurs at each stage of the procedure. The process to set up a pipe from a forked process into the parent process is as follows:

1. Call **.pipe**; save the returned input and output channels.

2. Call **.chdup** to duplicate channel 1, the standard output.

3. Call **.close** to close the standard output.

4. Call **.chdup** to make a duplicate of the output channel.

5. Fork the process using **.fexec** or **.fshell**.

6. Call **.close** to close channel 1, made in step 4.

7. Call **.chdup** to restore the standard output, temporarily saved in the standard output duplicate made in step 2.

8. Call **.close** to close the standard output duplicate made in step 2.

9. Call **.close** to close the output channel saved from step one.

The pipe is complete; any output the child process writes to its standard output (channel 1) can be read by the parent from the pipe's sending end (PIPEIN). For the sake of simplicity, assume that the parent process had only channels 0, 1, and 2 open at the outset. The next channel available for the operating system to allocate is now channel 3.

Here is a step by step examination of the procedure outlined above in steps 1 through 9.

1. The **.pipe** system call is made to set up a pipe. It returns two channels: a pipe input channel, and a pipe output channel which are 4 and 3, respectively. These channel numbers are allocated because of rule 2 stating that the lowest available channel numbers are allocated when a request is made. The channel numbers for input and output are saved as PIPEIN and PIPEOUT. The next available channel is number 5.

2. The channel duplication system call (**.chdup**) is used to duplicate standard output (channel 1). The duplicate channel is saved as stdoutdup, and is assigned channel number 5. Channel 5 is a duplicate of the standard output and is **connected** to the terminal display. The next available channel number is 6.

3. Standard output (channel 1) is closed using the system call **.close.** Channel 1 is now the lowest available channel number.

4. The pipe output channel (channel 3) is duplicated using the **.chdup** system call. Channel 1 is allocated as the duplicate channel.

5. The child process is forked using **.fshell** or **.fexec.** The child inherits channels 0, 1, and 2 from the parent process. At this point, the standard output of the child is written to the output channel which provides data to the pipe. Whenever the child sends output to the standard output channel, the output goes to the pipe's receiving end.

   The following steps are required to return the I/O channels of the parent process to their normal status.

6. Channel 1 is closed, making channel 1 the lowest available channel.

7. The **.chdup** system call is used to make a duplicate of STDOUTDUP, the duplicate of stdout made in step 2. Channel 1 is used for this purpose, and this channel is consequently returned to its original function as standard output channel for the parent.

8. STDOUTDUP and PIPEOUT channels are closed with the **.close** system call. At this point, the parent has its original standard input, output, and error channels in their normal status, plus an additional channel, the channel we saved in the variable PIPEOUT. The parent may read data generated by the child process from the pipe's output. When the pipe is no longer needed, close it using the **.close** system call. If the parent process terminates and the child attempts to write a byte to the pipe, the operating system sends the child a SIGPIPE signal and the child terminates.

The following programs illustrate this procedure.

```
#include <stdio.h>
#define  STDOUT 1

main(argc,argv)
int argc; char *argv[];
{
int pipein,pipeout,stdoutdup,pid;
char ch;

  struct command {      char *sh,  *cc,  *string; };  /* Structure for fshell*/

  struct command arglist = { "sh", "-c", ""};         /* Initialized to      */
                                                      /* "sh -c ..."         */

  if (argc != 2 ) printf("Wrong number of arguments \n");
  else  {                             /* Correct number of arguments        */
  arglist.string = *++argv;           /* Argument 1 from the command line
                                      is added to the arglist to be forked */


  /********  Begin setting up pipe  *********/

  pipe(&pipeout,&pipein);        /* Step 1. Get a pipe from Cromix, channels for
                                 saving pipe input & output channels in PIPEIN
                                 & PIPEOUT */

  stdoutdup = chdup(STDOUT);     /* Step 2. Make a duplicate of stdout channel*/

  close(STDOUT);                 /* Step 3. Close stdout. Channel 1 becomes next
                                 available channel because stdout(1) is closed*/

  chdup(pipeout);                /* Step 4. Make a duplicate of the pipe output
                                 channel.  Channel 1 would be assigned as dup */

  pid = fshell(arglist,0xb,0);   /* Step 5. Fork the child process. Child
                                 inherits channels 0,1, and 2.  Channel 1 -
                                 the standard output is duplicate of PIPEOUT  */

  close(STDOUT);                 /* Step 6. Close channel 1 that is currently
                                 linked to PIPEOUT.                           */

  chdup(stdoutdup);              /* Step 7. Duplicate stdoutdup.  returns
                                 channel 1, because it was closed above       */

  close(stdoutdup);              /* Step 8. Close stdoutdup and PIPEOUT       */
  close(pipeout);                /* because they aren't needed anymore        */

  /********  Done setting up the pipe  *********/


  /* Loop, reading a char from the pipe and printing the char in upper case */

  while((read(pipein,&ch,1)) == 1   ) putchar(toupper(ch));

      /*  Loop terminates when READ returns an end of file */

  kill(pid,3);                   /* Kill the forked process */

  }                              /* End if statement       */
}                                /* End Main               */
```

system call:	**.PRINTF**		
number:	1Bh		
purpose:	This call generates formatted output.		
user access:	all users		
summary:	b	=	channel
	hl	->	control string
	push		all arguments
	jsys		.printf
	pop		all arguments
calling parameters:	b	The b register contains the output channel number.	
	hl	The hl register pair points to the null terminated control string.	
	stack	All arguments to the printf call must be pushed onto the stack before the call and popped off of the stack after the call.	
return parameters:	none		
possible errors:			

The **.printf** call generates output that is a formatted string.

The null terminated control string is composed of regular characters and conversion specifications. Regular characters are copied directly to the output file. Conversion specification characters are introduced by the percent (%) sign and terminated by the conversion character itself.

The conversion specification characters have the following format:

%-xxx.yyyLz

The percent sign and the conversion character itself (**z**) are required, and all of the conversion specification characters in between are optional.

A minus sign may follow the percent sign. If it is included, the argument is left justified. Otherwise the argument is right justified.

Following this may be two strings of digits separated by a period (represented by **xxx.yyy**). The first of these numbers represents the minimum field width. If it is not included, the minimum field width is assumed to be zero. The second of these numbers represents the maximum field width. If it is not included, the maximum field width is as large as necessary.

If the character **L** appears after this, it signifies that the argument is a long (32-bit) number. If it is absent, the argument is assumed to be short (16 bits).

The conversion character itself (represented by **z**) may be any one of the following:

d	The argument is converted to a decimal number.
u	The argument is converted to an unsigned decimal number.
x	The argument is converted to an unsigned hexadecimal number.
c	The argument is assumed to be a single character. When this argument is pushed onto the stack, the character must be in the low order byte of the pushed register pair.
s	The argument is assumed to be a character string. A pointer to this string must be pushed onto the stack in place of the string itself.
l	The argument is a 32-bit integer.
,	If a comma appears after the percent sign in a decimal conversion, a comma appears in the output (as in 1,000,000).

Example:

PRINT FORMATTED OUTPUT system call (jsys .printf)

```
;
;
; Print the specified data pushed on the stack using the
; printf control string specified by the HL register pair.
;
;
;
;
                ld      b,stdout        ; standard output channel for console
                ld      hl,123          ; a number to print
                push    hl              ; push on stack
                ld      hl,ctrl_string  ; pointer to control string

                jsys    .printf         ; print formatted output system call

; Registers returned:
;       none

                pop     hl              ; restore stack


ctrl_string defb' \nThe number = %d\n',0        ; control string for printf


;       ************    end of example  ************
```

system call:	**.RDBYTE**
number:	16h
purpose:	This call reads a byte.
user access:	all users
summary:	b = channel jsys .rdbyte a = byte
calling parameters:	b The b register contains the channel number of the file.
return parameters:	a The a register contains the byte read.
possible errors:	?notopen ?filaccess ?ioerror ?endfile

The **.rdbyte** call reads the next sequential byte going toward the end of the file from the open file on the channel specified.

```
system call:    .RDLINE
     number:    18h
    purpose:    This call reads a line.

user access:    all users

    summary:    b      =    channel
                de     =    maximum bytes
                hl     ->   buffer
                jsys       .rdline
                de     =    bytes read
```

```
    calling
 parameters:    b     The b register contains the channel
                      number of the file.

                de    The de register pair contains the
                      maximum number of bytes to be read
                      by this call.

                hl    The hl register pair points to the
                      buffer in which the line is
                      returned.


     return
 parameters:    de    The de register pair contains the
                      actual number of bytes read,
                      including the line terminator.

   possible
    errors:     ?notopen
                ?filaccess
                ?ioerror
                ?endfile
```

The **.rdline** call reads a line, or a number of sequential bytes moving toward the end of the file, from the file open on the specified channel.

The buffer is filled with bytes until an end of line indicator is encountered - a linefeed or null character.

Example:

READ LINE system call (jsys .rdline)

```
;
;
; Read a line from the channel specified by the B register.
;
;
;
;
;

            ld      de,50           ; byte count (number of bytes to read)
            ld      hl,buffer       ; pointer to 50 byte line buffer
            ld      b,stdin         ; standard input channel for console

            jsys    .rdline         ; read line system call

; Registers returned:
;       DE = bytes read


buffer:         defs    50              ; line buffer


;       ************    end of example   ************
```

system call:	**.RDSEQ**
number:	14h
purpose:	This call reads the specified number of bytes.
user access:	all users

summary:

```
b     =   channel
de    =   byte count
hl   ->   buffer
jsys     .rdseq
de    =   number of bytes read
```

calling parameters:

b — The b register contains the channel number associated with the file to be read.

de — The de register pair contains the number of sequential bytes to be read starting from the current position of the file pointer.

hl — The hl register pair points to the buffer where bytes are returned.

return parameters:

de — The de register contains the actual number of bytes read.

possible errors:

?notopen
?filaccess
?ioerror
?endfile

The **.rdseq** call reads the next specified number of bytes, moving toward the end of the file, from the file open on the specified channel.

Example:

READ SEQUENTIAL system call (jsys .rdseq)

```
;
;         It is assumed that a channel was opened previously
;         and the channel number is in the B register
;         (see OPEN system call).
;
; Read sequentially from the channel specified by the B register to the
; buffer specified by the HL register pair.
;

          ld   de,200              ; byte count(number of bytes to read)
          ld   hl,buffer           ; pointer to buffer

          jsys .rdseq              ; read sequential system call

; Registers returned:
;         de = bytes read



buffer:   defs 200                 ; 200 byte buffer


;         ************     end of example     ************
```

system call:	**.SETDATE**
number:	31h
purpose:	This call changes the date.
user access:	privileged user
summary:	e = year h = month l = day of the month jsys .setdate
calling parameters:	e The e register contains the year minus 1900. This means that 1980 is represented as 80 and 2004 is 104. h The h register contains the month where 1 represents January, 2 represents February, etc. l The l register contains the day of the month in the range between 1 and 31 inclusive.
return parameters:	none
possible errors:	

The **.setdate** call changes the Cromix system clock to the date specified. The parameters are binary numbers.

Example:

SET DATE system call (jsys .setdate)

```
;
;
; Set the operating system date.
;
;
;
;
                ld      e,81            ; year
                ld      h,5             ; month
                ld      l,23            ; date

                jsys    .setdate        ; set date system call

; Registers returned:
;       none



;       ************    end of example  ************
```

system call:	**.SETDIR**
number:	03h
purpose:	This call changes the current directory.
user access:	all users
summary:	hl -> pathname jsys .setdir
calling parameters:	hl The hl register pair points to the new directory pathname. The pathname must be terminated by a null character.
return parameters:	none
possible errors:	?notdir ?diraccess

The **.setdir** call changes the current directory to the one specified.

Example:

SET DIRECTORY system call (jsys .setdir)

```
;
;
; Select the directory specified by the HL register pair.
;
;
;
;
                ld      hl,path_name    ; pointer to directory pathname

                jsys    .setdir         ; set directory system call

; Registers returned:
;       none



path_name:      defb    '/usr/my_test_directory',0      ; directory name


;       ************    end of example  ************
```

system call: **.SETGROUP**
number: 37h
purpose: This call changes the group id.

user access: all users

summary:

```
b     =   type of id to change
c     =   new id label
hl    =   new id number
jsys    .setgroup
```

calling parameters:

b — The b register contains the type of id to be changed.

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM

c — The c register indicates the value of the id type specified by the b register. This value may be the value of one of the other id types or the value specified by the hl register.

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM
ID.HL

hl — If the c register contains ID.HL, the hl register pair must contain a 16 bit id number.

return parameters: none

possible errors:

The **.setgroup** call changes the group id of the current process to the one specified. This call may be invoked only by a privileged user when the c register has the value of ID.HL.

Example:

SET GROUP system call (jsys .setgroup)

```
;
;
; Change the current login group id to 4.
;
;
;
;

            ld      b,ID.LOGIN          ; type of id to change
            ld      c,ID.HL             ; change to value of hl register pair
            ld      hl,4                ; new value = group 4

            jsys    .setgroup           ; set group system call

; Registers returned:
;       none



;       ************    end of example   ************
```

system call:	**.SETMODE**
number:	13h
purpose:	This call changes the characteristics of a character device.
user access:	all users

summary:

```
b     =   channel
c     =   mode type
d     =   new value
e     =   mask
jsys     .setmode
d     =   old value
```

calling parameters:

b The b register contains the channel number of the opened device.

c The c register contains the MODE TYPE to be set. The c register may be loaded with one of the mode types listed below.

d The d register contains the new value of the mode type specified by the c register. Refer to the table below.

e The e register, in MD_MODE1, MD_MODE2, and MD_MODE3, is a mask indicating which characteristics to change.

Mode types for printers and terminals:

C Register	Significance
MD_ISPEED	input speed (baud rate)
MD_OSPEED	output speed (baud rate)
MD_MODE1	flags: tandem, xtab, lcase, echo, crdevice, raw, odd, even
MD_MODED	delays: nldelay, tabdelay, crdelay, ffdelay, bsdelay
MD_MODE2	flags: pause,

	notimmecho, noecnl, sgenable, abenable, xff, wrap, sigallc
MD_MODE3	flags: escretn, fnkeys, hupenab, sighupall, cbreak, binary, discard
MD_ERASE	auxiliary input erase character
MD_DELECHO	erasure echo character
MD_LKILL	input line kill character
MD_USIGNAL	SIGUSER signal key
MD_LENGTH	page length
MD_WIDTH	page width
MD_BMARGIN	bottom margin
MD_IFLUSH	flush input buffers
MD_FNKEYS	enable or disable 3102 function keys
MD_STATUS	input buffer status.

return parameters: d — The d register contains the previous value of the mode type specified by the c register.

possible errors: ?badvalue

The **.setmode** call changes the characteristics of a character device. Refer also to the **.getmode** system call and Mode utility.

MD_ISPEED & MD_OSPEED setmode call

If the c register contains MD_ISPEED or MD_OSPEED, the d register must be loaded with the desired input or output baud rate using the speed codes below.

Speed Code	Baud rate
S_110	110 baud
S_150	150 baud
S_300	300 baud
S_1200	1200 baud
S_2400	2400 baud
S_4800	4800 baud
S_9600	9600 baud
S_19200	19200 baud
S_NOCHG	no change of baud rate
S_CTSWAIT	wait for Clear to Send
S_HANGUP	hang up phone when device closed
^Sfl_AUTO	input CRs form the keyboard to set baud rate

MD_MODE1 setmode call

If the c register contains MD_MODE1, the e register is a mask and the d register is the new value. For example, if the XTAB bit in the e register is set, the corresponding bit in the d register indicates the new value of XTAB (0 = disabled, 1 = enabled).

Bit in D	Significance
TANDEM	send XOFF/XON to control filling of input buffer
XTAB	expand tabs
LCASE	convert input to lower case
ECHO	echo (full duplex) input
CRDEVICE	carriage return device
RAW	(see RAW Table below)
ODD	(see Parity Table below)
EVEN	(see Parity Table below)
HUPENAB	hang modem up when device closed
SIGHUPALL	send SIGHANGUP signals to all processes using the TTY device if modem hangs up
CBREAK	see RAW table below
BINARY	see RAW table below

PARITY TABLE

The two bits, ODD and EVEN, are combined to produce four combinations. These are listed in the following table (where + means enabled and - means disabled).

EVEN	ODD	Function for Input Characters
-	-	does not check parity but strips parity bit
+	-	checks for even parity before stripping parity bit
-	+	checks for odd parity before stripping parity bit
+	+	leaves parity unchecked and unchanged
EVEN	**ODD**	**Function for Output Characters**
-	-	strips parity bit
+	-	makes character have even parity
-	+	makes character have odd parity
+	+	leaves parity bit unchanged

MD_MODED setmode call

If the c register contains MD_MODED, the e register indicates the delay to be set and the d register contains the new value. For example, if the NLDELAY bits in the e register are set, the byte in the d register indicates the new value of NLDELAY.

Bit in D	Significance
NLDELAY	newline delay
TABDELAY	tab delay
CRDELAY	carriage return delay
FFDELAY	formfeed delay
BSDELAY	backspace delay

DELAY TABLE

Character	DELAYcode Bits	QTTY Values (seconds)	TTY Values (nulls)
newline	0 and 1	0, .1, .2, .3	0, 4, 8, 12
TAB	2 and 3	0, .1, .2, .3	0, 4, 8, 12
RETURN	4 and 5	0, .1, .2, .3	0, 4, 8, 12
formfeed	6	0, .8	0, 128
backspace	7	0, .1	0, 4

MD_MODE2 setmode call

If the c register contains MD_MODE2, the e register is a mask and the d register is the new value. For example, if the PAUSE bit in the e register is set, the corresponding bit in the d register indicates the new value of PAUSE (0 = disabled, 1 = enabled).

Bit in D	Significance
PAUSE	after MD_LENGTH number of lines, wait for CNTRL-Q to continue
NOTIMMECHO	do not echo characters typed ahead
NOECNL	no echoing of line terminators
ABENABLE	send SIGABORT signal if CNTRL-C is pressed
XFF	expands form feeds as NEWLINEs
WRAP	software wrap around and inserts NEWLINE when page width is exceeded
SGENABLE	send SIGUSER signal if MD_USIGNAL key is pressed (See discussion below)
SIGALLCHARS	send SIGUSER signal for every key pressed (See discussion below.)

SIGENABLE, SIGCHAR, and SIGALLCHARS

If SIGENABLE is on and SIGALLCHARS is off, pressing the SIGCHAR key causes terminal devices TTY, QTTY, and MTTY to send a SIGUSER signal to all processes controlled by the terminal. The SIGCHAR key character is not put into the input stream. If SIGENABLE is off, then the SIGCHAR key is treated in the same manner as any other key. The terminal which controls a process is the terminal on which the owner of the process logged on to the system.

If SIGENABLE and SIGALLCHARS are both on, pressing the SIGCHAR key causes the SIGUSER signal to be sent to all processes controlled by the terminal, but the SIGCHAR key character is also put into the input stream.

If SIGALLCHARS is on but SIGENABLE is off, every terminal keystroke pressed before a system call to read input has been made sends the SIGUSER signal to all controlled processes. (Only characters typed-ahead send signals.) The characters are also put into the input stream.

Note that Shells are set up to ignore SIGUSER signals, so that a user is not logged off by them. Any program running in a nondetached mode that does not either ignore or trap SIGUSER signals is aborted by them. (The **.signal** system call provides a means for ignoring or trapping signals.)

MD_MODE3 setmode call

If the c register contains MD_MODE3, the e register is a mask and the d register is the new value. For example, if the ESCRETN bit in the e register is set, the

corresponding bit in the d register indicates the new value of ESCRETN (0 = disabled, 1 = enabled).

ESCRETN

If ESCRETN is enabled, the ESC key acts as a line terminator.

FNkEYS

If FNKEYS is enabled, the terminal drivers TTY, QTTY, and MTTY perform the handshaking that the Cromemco 3102 terminal expects whenever a function key is pressed. (The driver echoes a CNTRL-B for each of the two bytes the terminal sends.) This allows the 2-byte function key sequences of the 3102 to be transmitted to a program when they are pressed.

DISCARD

When a driver is first used, a data area is allocated where its parameters (including its mode characteristics) are saved. This data area is reserved for the driver until it is DISCARDed. For most drivers, the location of the data area depends on the port address of the interface board used. For example, terminal TTY2 and serial line printer SLPT2 both use the TU-ART interface board addressed at 20h. For this reason, after access to TTY2 is obtained, SLPT2 cannot be opened until the driver for TTY2 has first been discarded. If the command **mode tty2 discard** is given, the data space for TTY2 is discarded as soon as the device TTY2 is closed. Then SLPT2 can be opened.

HUPENABLE

If this switch is on and an IOP terminal device, a QTTY or an MTTY, closes, the modem on the IOP device is hung up.

SIGHUPALL

If this switch is on and the modem of an IOP terminal device, QTTY or MTTY, hangs up, the signal SIGHANGUP is sent to all processes controlled by the device. A process is controlled by the terminal with which the user who initiated the process logged in. For example, a user who has logged in on MTTY1 and hangs up without

logging off is logged off by the resulting SIGHANGUP signal, provided SIGHUPALL is enabled.

RAW TABLE

If either CBREAK or BINARY of MD_MODE3 is set, or if RAW of MD_MODE1 is set, any read from the device will return after each input character. These parameters also serve to disable the action of various other parameters. These effects are listed in the table below. A + means that the parameter causes the given effect, a space means that it does not.

Effect	CBREAK	RAW	BINARY
Return after each character input	+	+	+
No erase, linekill, or EOF (CNTRL-Z) functions	+	+	+
No output PAuse or output Width truncation		+	+
Treat XOFF (CNTRL-S), XON (CNTRL-Q) as regular input		+	+
No tandem mode (i.e., no input buffer flow control)			+
Treat CNTRL-C and SIGCHAR key as regular input			+
No checking or changing of parity bit			+
No delays after any output control chars such as tabs			+
No echoing of input			+
No function key decoding			+
No character transformations (i.e., ignore the LCase, CRDEVice, and TABexpand settings)			+

MD_ERASE setmode call

If the c register contains MD_ERASE, the d register is the new value for the erase character.

MD_DELECHO setmode call

If the c register contains MD_DELECHO, the d register is the new value for the deletion echo character.

MD_LKILL setmode call

If the c register contains MD_LKILL, the d register is the new value for the line kill character.

MD_USIGNAL setmode call

If the c register contains MD_USIGNAL, the d register is the new value for the user signal character.

MD_LENGTH setmode call

If the c register contains MD_LENGTH, the d register is the new value for the page length.

MD_WIDTH setmode call

If the c register contains MD_WIDTH, the d register is the new value for the page width.

MD_BMARGIN setmode call

If the c register contains MD_BMARGIN, the d register is the new value for the bottom margin.

MD_IFLUSH setmode call

If the c register contains MD_FLUSH, all input buffers are flushed.

MD_FNKEYS setmode call

If the c register contains MD_FNKEYS, the d register contains either 1 to enable the function keys or 0 to disable them.

Example:

SET MODE system call (jsys .setmode)

```
;
; Change the mode of the current console to the following:
;
;         expand tabs - off
;         echo - on
;
                ld      b,stdout         ; current console channel number
                ld      c,MD_MODE1       ; mode type to change
                ld      d,^xtab|^echo    ; load mask with options to change
                ld      e,^echo          ; new values, echo=on xtab=off

                jsys    .setmode         ; setmode system call

; Registers returned:
;         none


;         ************    end of example    ************
```

system call:	**.SETPOS**
number:	11h
purpose:	This call changes the position of a file pointer.
user access:	all users
summary:	b = channel c = mode dehl = file pointer jsys .setpos

calling parameters:

b — The b register contains the channel number of an open file.

c — The c register contains the mode. This is the location from and direction to which the file pointer position is established.

FWD.BEGIN	forward from the beginning of file
FWD.CURRENT	forward from the current position
FWD.END	forward past the end of file
BAK.CURRENT	backward from the current position
BAK.END	backward from the end of file

dehl — The de and hl register pairs contain the position change of the file pointer.

return parameters: none

possible errors:

?notblk
?filaccess
?notopen

The **.setpos** call changes the file pointer position to the specified logical byte position.

Example:

SET FILE POSITION system call (jsys .setpos)

```
;
;       It is assumed that a channel was previously opened
;       and the channel number is in the B register.
;       (see OPEN system call)
;
; Set the file pointer (specified by the HL register pair) 2250 bytes
; from the beginning of the file specified by the B register.
;
;

                ld      de,0            ; position 2250
                ld      hl,2250         ; /
                ld      c,FWD.BEGIN     ; from beginning of file

                jsys    .setpos         ; set position system call

; Registers returned:
;       none




;       ************    end of example   ************
```

system call:	**.SETPRIOR**
number:	39h
purpose:	This call changes the process priority.
user access:	all users (priorities 0 to 40) privileged user (priorities -40 to 40)
summary:	1 = priority number jsys .setprior
calling parameters:	1 The 1 register contains the new priority number (-40 to 40).
return parameters:	none
possible errors:	

The **.setprior** call changes the current process priority as specified by the 1 register. The priority number must be in the range of -40 (the highest priority) to 40. Only a privileged process may set a priority in the range of -40 to -1. The default priority assigned by the operating system is 0.

system call: **.SETTIME**
number: 33h
purpose: This call changes the time.

user access: privileged user

summary:

```
e     =   hour
h     =   minute
l     =   second
jsys     .settime
```

calling parameters:

e — The e register contains the hours portion of the current time based on a 24-hour clock (e.g., 6 p.m. is represented as 18 hours).

h — The h register contains the minutes portion of the current time. This is the number of minutes since the current hour started.

l — The l register contains the seconds portion of the current time. This is the number of seconds since the current minute started.

return parameters: none

possible errors:

The **.settime** call changes the Cromix system clock to the time specified. The parameters are binary numbers.

Example:

SET TIME system call (jsys .settime)

```
;
;
; Set the operating system time.
;
;
;
;

                ld      e,11            ; hour
                ld      h,30            ; minute
                ld      l,29            ; second

                jsys    .settime        ; set time system call

; Registers returned:
;       none



;       ************    end of example   ************
```

system call: **.SETUSER**
number: 35h
purpose: This call changes the user id.

user access: all users

summary:

```
b     =   type of id to change
c     =   new id type
hl    =   new id number
jsys     .setuser
```

calling parameters:

b — The b register contains the type of id that is to be changed.

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM

c — The c register is used to indicate the value of the id type specified by the b register. This value may be the value of one of the other id types or the value specified by the hl register.

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM
ID.HL

hl — If the c register contains ID.HL, then the hl register pair must contain a 16 bit id number.

return parameters: none

possible errors:

The **.setuser** call changes the id of the current process to that which is specified. This call may be invoked only by a privileged user when the c register has the value of ID.HL.

Example:

SET USER system call (jsys .setuser)

```
;
;
; Change the current user id to 2.
;
;
;
;

        ld      b,ID.LOGIN      ; request login id type
        ld      c,ID.HL         ; new user value in hl register pair
        ld      hl,2            ; new value = user 2

        jsys    .setuser        ; set user system call

; Registers returned:
;       none



;       ************    end of example  ************
```

system call: **.SHELL**
number: 49h
purpose: This call initiates a Shell process.

user access: all users

summary:
```
de      ->  argument list
jsys        .shell
```

calling parameters: de — The de register pair points to a list of 16-bit pointers. The list of pointers is terminated by a null (0) pointer. Each pointer points to a null terminated character string. Each string is an argument passed to the forked process.

return parameters: none

possible errors:

The **.shell** call initiates and assumes execution of a Shell process. A new PID is not generated.

Options

These options are needed only when a program is calling a Shell. They are not useful when a Shell is called from the terminal.

The **-c** option indicates that the command line being passed to the Shell has not been parsed.

The **-p** option indicates that the command line being passed to the Shell has been parsed.

The **-q** option requests that lines from a command file not be echoed to the terminal.

Notes

The **.shell** call expects arguments to be in one of the following three forms:

Form 1 (passing command filenames)

```
de ->   arg 0  ->  "shell \0"
        arg 1  ->  arg 1 (a command filename)
        arg 2  ->  arg 2
        .
        .
        .
        0
```

Form 2 (passing a parsed argument list)

```
de ->   arg 0  ->  "shell \0"
        arg 1  ->  -p \0"
        arg 2  ->  command name (terminated by \0)
        arg 3  ->  command's first argument (terminated by \0)
        arg 4  ->  command's second argument (terminated by \0)
        .
        .
        .
        .
        0
```

Form 3 (passing a command line)

```
de ->   arg 0  ->  "shell \0"
        arg 1  ->  "-c \0"
        arg 2  ->  command line (terminated by \0)
        0
```

Example:

SHELL system call (jsys .shell)

```
;
; The operating system creates a new process
; and executes the command file specified by the command
; argument. The specified arguments (one and two) are
; also passed to the new process.
;

                ld      de,arg_list      ; pointer to argument list

                jsys    .shell           ; shell system call


; Registers returned:
;       none


arg_list:       defw    arg_0            ; pointer to argument 0
                defw    arg_1            ; pointer to argument 1
                defw    arg_2            ; pointer to argument 2
                defw    0                ; end of argument pointer list


arg_0           defb    'sh',0                      ; request a shell
arg_1           defb    '-c',0                      ; do not parse
                                                    ; arguments
arg_2           defb    'mode prt > a_test_file',0  ; program name
                                                    ; and arguments


;       ************    end of example  ************
```

system call:	**.SIGNAL**
number:	40h
purpose:	This call sets up a process to receive a signal.
user access:	all users

summary:

```
c      =   type of signal
hl     =   execution address
jsys       .signal
hl     =   previous execution address
```

calling parameters:

c — The c register contains the type of signal.

SIGABORT	CNTRL-C signal
SIGUSER	user specifiable key signal
SIGKILL	kill signal
SIGTERM	terminate signal
SIGALARM	alarm signal
SIGPIPE	broken pipe signal
SIGHANGUP	modem hangup signal

hl — The hl register pair contains the program address to which control is transferred. If the hl register pair contains 0000, the process aborts upon receipt of the specified signal; if hl contains 0001, the signal is ignored.

return parameters:

hl — The hl register pair contains the previous execution address.

possible errors:

?badcall
?signal

If the **.signal** call has been used to set up a subroutine address, control is passed to the subroutine at the address specified when the signal is received. The program returns to the point of execution where the signal was received on encountering a RET instruction. Further signals of the same kind will then be ignored unless **.signal** is used to set up the address again.

```
; This program sets up to receive 2 signals.
; One is the sigabort signal (user CNTRL-C) and the
; other is the sigalarm signal. After the set up,
; alarm is called with the value of 10 which
; gives a sigalarm signal after 10 seconds. If this
; signal is received, the program  prints an alarm message and exits
; to the operating system. If the user enters a CNTRL-C, the program
; prints a CNTRL-C message and aborts.
;


*include        jsysequ.z80             ; standard system call equate
;
sig_test:
;{
                ld      sp,stack        ; load the stack pointer

                ld      c,sigalarm      ; set up for an alarm signal
                ld      hl,time_out     ; address of time out routine
                jsys    .signal         ; signal system call

                                        ; check for system call error
                s  jp   c,error         ;       Yes, then goto error routine
                                        ; else
                                        ;       continue

                ld      c,sigabort      ; set up for an abort signal
                ld      hl,abort_trap   ; address of abort trap routine
                jsys    .signal         ; signal system call

                                        ; check for system call error
                s  jp   c,error         ;       Yes, then goto error routine
                                        ; else
                                        ;       continue


                ld      hl,10           ; request alarm signal in 10 second
                jsys    .alarm          ; alarm system call

                                        ; check for system call error
                s  jp   c,error         ;       Yes, then goto error routine
                                        ; else
                                        ;       continue

loop:           jp      loop            ; loop

;}

time_out:
;{
                ld      b,stdout        ; print time out message on console
```

```
                ld      hl,ctrl1        ; printf control string
                jsys    .printf         ; printf system call

                ld      hl,-1           ; indicate an error condition
                jsys    .exit           ; and exit to the operating system
;}

abort_trap:
;{
                ld      b,stdout        ; print abort message on console
                ld      hl,ctrl2        ; printf control string
                jsys    .printf         ; printf system call

                ld      hl,0            ; indicate no error
                jsys    .exit           ; and return to the operating system
;}

error:
;{
                ld      b,stderr        ; standard error channel for console
                jsys    .error          ; error system call
                ld      hl,-1           ; indicate errors
                jsys    .exit           ; and exit to operating system
;}

; printf control strings and stack area

ctrl1:          defb    '\nAlarm abort\n',0
ctrl2:          defb    '\nCtrl C abort\n',0

                defs    10              ; stack area = 10 bytes
stack:          equ     $               ; starting address of stack

                end     sig_test
```

system call:	**.SLEEP**
number:	42h
purpose:	This call puts a process to sleep.
user access:	all users
summary:	hl = number of seconds to sleep jsys .sleep hl = number of seconds left to sleep
calling parameters:	hl The hl register pair contains the time in seconds for which the process is to sleep.
return parameters:	hl The hl register pair returns the number of seconds left if the entire allotted time was not expended before the process was aborted.
possible errors:	

The **.sleep** system call is used to put a process to sleep for a specified interval in seconds. This frees processor time for other processes.

system call:	**.TRUNC**
number:	0Dh
purpose:	This call truncates an open file.
user access:	all users
summary:	b = channel jsys .trunc
calling parameters:	b The b register contains the channel number of the open file.
return parameters:	none
possible errors:	?notopen

The **.trunc** system call deletes the file from the current file pointer position through the end of the file.

system call:	**.UNLOCK**
number:	3Fh
purpose:	This call is used to unlock a locking sequence.
user access:	all users
summary:	c = lock type de = lock length hl -> lock sequence jsys .unlock
calling parameters:	c The c register must contain the same value as it contained when the corresponding **.lock** system call was executed. de The de register pair must contain the same value as it contained when the corresponding **.lock** system call was executed. hl The hl register pair must contain the same value as it contained when the corresponding **.lock** system call was executed.
return parameters:	none
possible errors:	

The **.unlock** system call unlocks a locking sequence that was initiated by the **.lock** system call. Please refer to the **.lock** system call for more information.

system call:	**.UNMOUNT**
number:	05h
purpose:	This call disables access to a file system.
user access:	privileged user
summary:	c = eject flag hl -> block device pathname jsys .unmount
calling parameters:	c If the c register contains a 1, the diskette that is unmounted is ejected. If c contains a 0, the diskette is not ejected. hl The hl register pair points to a buffer containing the pathname of the block device to be unmounted.
return parameters:	none
possible errors:	?notmount ?fsbusy ?badname ?notexist

The **.unmount** call, used in conjunction with **.mount**, declares that the device no longer has the previously specified file system.

When the system is unmounted, the file system pathname reverts to being a dummy pathname.

Example:

UNMOUNT FILE SYSTEM system call (jsys .unmount)

```
;
; Unmount the device specified by the HL register pair.
;
;
;
;
;
;               ld      c,1             ; eject disk flag
                ld      hl,path_device  ; pointer to device

                jsys    .unmount        ; unmount file system system call

; Registers returned:
;       none



path_device:    defb    '/dev/fda',0    ; large floppy drive A


;       ************    end of example  ************
```

system call:	**.UPDATE**
number:	52h
purpose:	This call updates all open files.
user access:	all users
summary:	jsys .update
calling parameters:	none
return parameters:	none
possible errors:	?ioerror

The **.update** call causes all open files to be updated with the current contents of their buffers. This is done automatically upon closing a file.

system call:	**.VERSION**
number:	55h
purpose:	This call returns the operating system version number.
user access:	all users
summary:	jsys .version hl = version number
calling parameters:	none
return parameters:	hl The hl register pair contains the Cromix Operating System version number.
possible errors:	none

The **.version** call returns the version number of the operating system.

Note

The version number in the hl register is encoded in BCD, with the integer portion in the h register and the fractional in the l register.

system call:	**.WAIT**		
number:	45h		
purpose:	This call waits for the termination of a child process.		
user access:	all users		
summary:	c	=	conditional flag
	hl	=	process id number
	jsys		.wait
	hl	=	child process number
	de	=	process termination status
	c	=	system termination status

calling parameters:

c — If the c register equals zero, the call will not return until a child process has terminated.

If the c register equals one, this call returns immediately. An error is returned if no child process has terminated.

hl — If the hl register pair contains a zero, this call waits for the termination of any child process.

If the hl register pair is set equal to a process id (PID) number, this call waits for the termination of the specified process.

return parameters:

hl — The hl register pair contains the child process number.

de — The de register pair contains the process termination status returned by the **.exit** system call.

c — If the c register equals 0, the child process is terminated through **.exit**. Otherwise the c register contains the number of the signal which caused its termination and the value in the de register is undefined.

possible
errors: ?nochild

The **.wait** call informs the parent process when a child process is no longer active.

system call: **.WRBYTE**
number: 17h
purpose: This call writes a byte.

user access: all users

summary:

```
a     =   byte
b     =   channel
jsys    .wrbyte
```

calling parameters:

a — The a register contains the byte to be written.

b — The b register contains the channel number of the file.

return parameters: none

possible errors:

?notopen
?filaccess
?ioerror

The **.wrbyte** call writes a byte to the file open on the specified channel. The byte is written just after the last byte which was written since the device was opened. Note that this can overwrite information written to the file when it was previously open.

system call:	**.WRLINE**
number:	19h
purpose:	This call writes a line.
user access:	all users

summary:

```
b      =    channel
hl     ->   buffer
jsys       .wrline
de     =    byte written
```

calling parameters:

b	The b register contains the channel number of the file.
hl	The hl register pair points to the buffer where the line to be written is stored.

return parameters:

de	The de register pair contains the number of bytes written.

possible errors:

?notopen
?filaccess
?ioerror

The **.wrline** call writes a line (a series of sequential bytes) to the device open on the specified channel. The bytes are written just after the last byte written since the device was opened.

Bytes are written until line terminator is encountered - a linefeed or newline. Note that this can overwrite information written to the file when it was previously open.

Example:

WRITE LINE system call (jsys .wrline)

```
;
; Write a line to the channel specified by the B register.
;
;
;
;
;
;

        ld      hl,buffer       ; pointer to line to write
        ld      b,stdin         ; standard output channel for console

        jsys    .wrline         ; write line system call

; Register returned:
;       DE = number of bytes written



buffer:         defb    'This is a test line\n'


;       ************    end of example  ************
```

system call: **.WRSEQ**
number: 15h
purpose: This call writes sequentially.

user access: all users

summary:

```
b     =   channel
de    =   byte count
hl    ->  buffer
jsys     .wrseq
de    =   bytes written
```

calling parameters:

b — The b register contains the channel number of the file.

de — The de register pair contains the number of sequential bytes to be written starting from the current position of the file pointer.

hl — The hl register pair points to the buffer in which the bytes to be written are stored.

return parameters:

de — The de register pair contain the actual number of bytes written. If this is not equal to the value of the calling parameter, an error is returned.

possible errors:

?notopen
?filaccess
?ioerror

The **.wrseq** call writes the next sequential specified number of bytes to the device open on the specified channel. The bytes are written just after the last byte written since the file was opened. Note that this can overwrite information written to the file when it was previously open.

Example:

WRITE SEQUENTIAL system call (jsys .wrseq)

```
;
;          It is assumed that a channel was previously opened
;          and the channel number is in the B register
;          (see OPEN system call).
;
; Write the content of the buffer to the channel specified by the
; B register.
;


         ld    de,200     ; byte count(number of bytes to write)
         ld    hl,buffer  ; pointer to buffer

         jsys .wrseq      ; write sequential system call

; Register returned:
;          de = number of bytes written



buff:    def 200          ; 200 byte buffer


;        ************     end of example     ************
```

Appendix A

SETTING UP - HARDWARE

This appendix describes how to set up the Cromix Operating System hardware. Please refer to Chapter 6 for software considerations, Chapter 6 and Appendix B for IOP/Quadart considerations.

MEMORY BOARDS

The Cromemco Cromix Operating System is designed to operate with from two to seven Cromemco 64KZ Random Access Memory Boards. One 64KZ RAM board resides in bank 0 and is reserved for the operating system. This board also resides in all unused banks.

A minimum system requires two 64KZ RAM boards. One of these is reserved for the operating system and the other is dedicated to executing user programs.

Each additional 64KZ RAM board allows an additional process to be executed concurrently.

Note that a system with two memory boards can support several users, but that **no two processes which require full banks of memory can be executed simultaneously.** A **Shell command** is **not** considered a process because it is executed within the system bank and does not require any user memory. Thus, a minimum system can execute a Shell command **while** a process is being executed. In addition, a minimum system can drive the printer through the system bank by means of the **Spool utility.** This allows a minimum system to print a file and execute a program at the same time.

The number of users, or, more accurately, the number of concurrent processes which a system supports, can be determined by consulting the following table. In some cases more than one process can execute in one bank of memory. (Refer to Chapter 10)

Number of 64KZ Boards	Number of Users
2	1
3	2
4	3
5	4
6	5
7	6

Amount of Memory vs. Number of Users

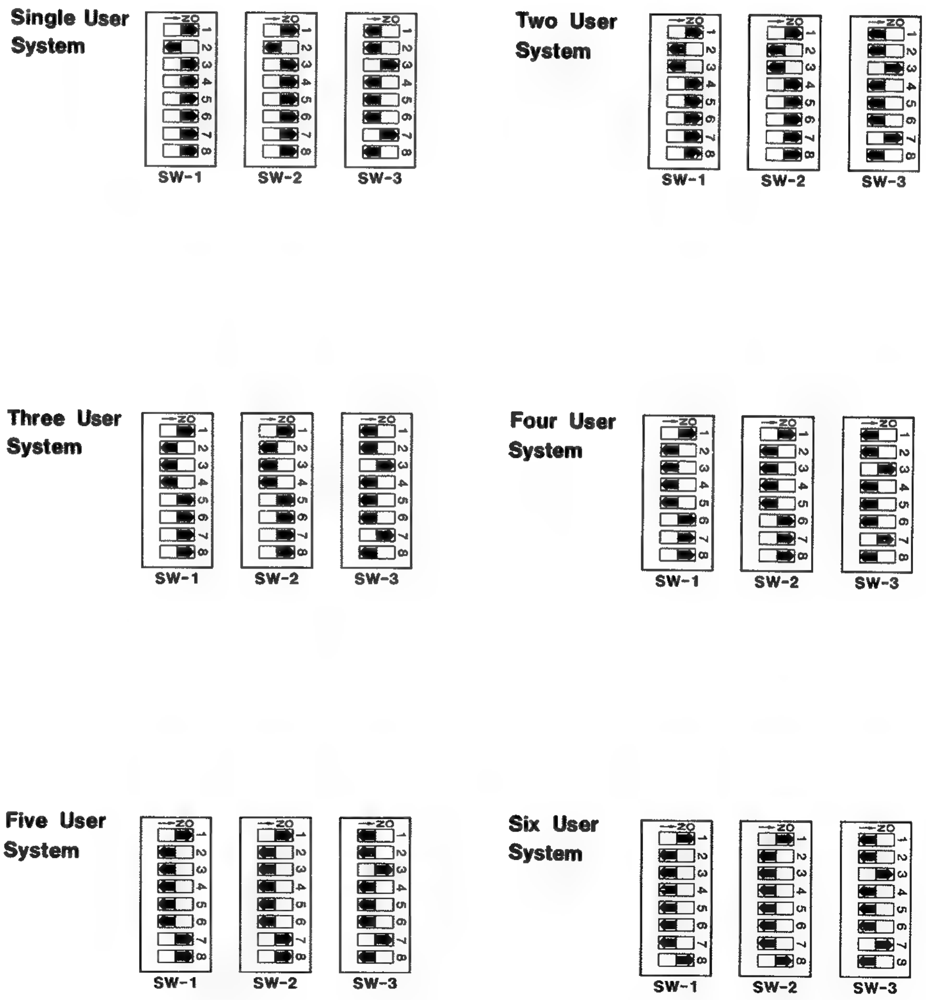
The switches on the 64KZ memory boards may be properly set by referring to the following two pages of 64KZ switch settings.

One board should be established as the system bank by setting the switches as indicated for the appropriate number of users. Refer to the diagrams entitled **64KZ System Bank Switch Settings. Only one board should be set according to the diagrams in this table.** The switches on additional 64KZ boards should be set in the manner described in the following paragraph.

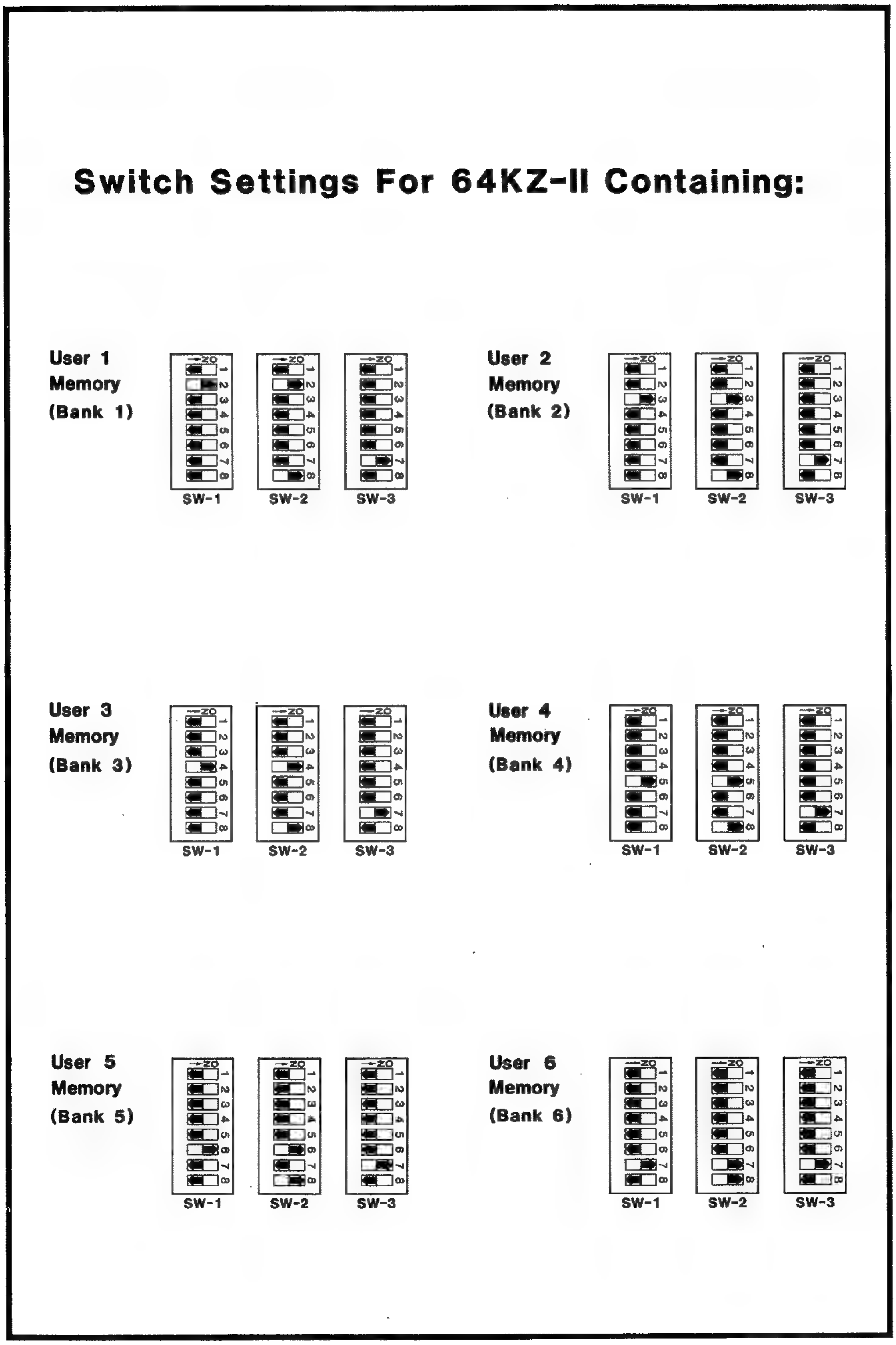
The other board(s) should be established as user bank(s) by setting the switches as indicated in the diagrams entitled **64KZ User Bank Switch Settings.** The switches on one board should be set as indicated by the diagram under the title **bank 1.** If there is a second board, it should be set according to the **bank 2** diagram. Additional banks should be established **in numerical order** for as many 64KZ boards used.

If additional memory is added to the system at a later time, it is important to remember to change the switch settings on the system (bank 0) board.

Switch Settings For 64KZ-II Containing The Cromix Operating System (Bank 0)



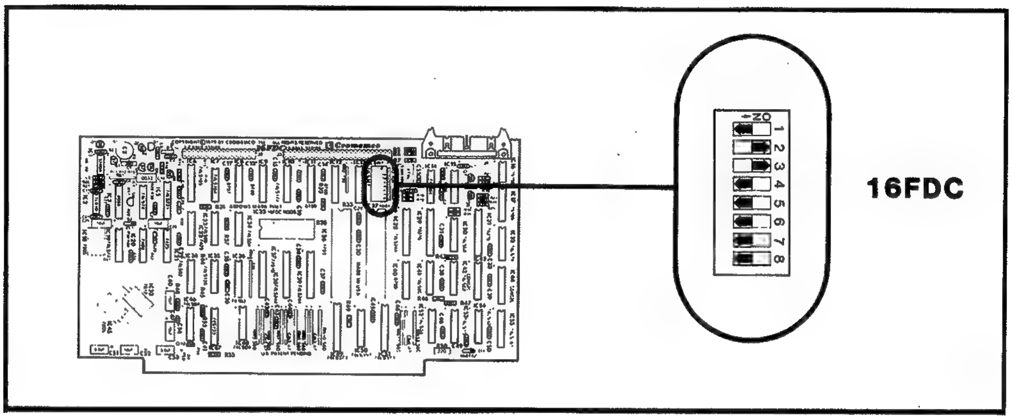
64KZ-II System Bank Switch Settings



64KZ-II User Bank Switch Settings

FLOPPY DISK CONTROLLER

The following switch settings are recommended for the 4FDC or 16FDC disk controller. Note that switch sections 5 through 8 only apply to the 16FDC.



4FDC & 16FDC Switch Settings

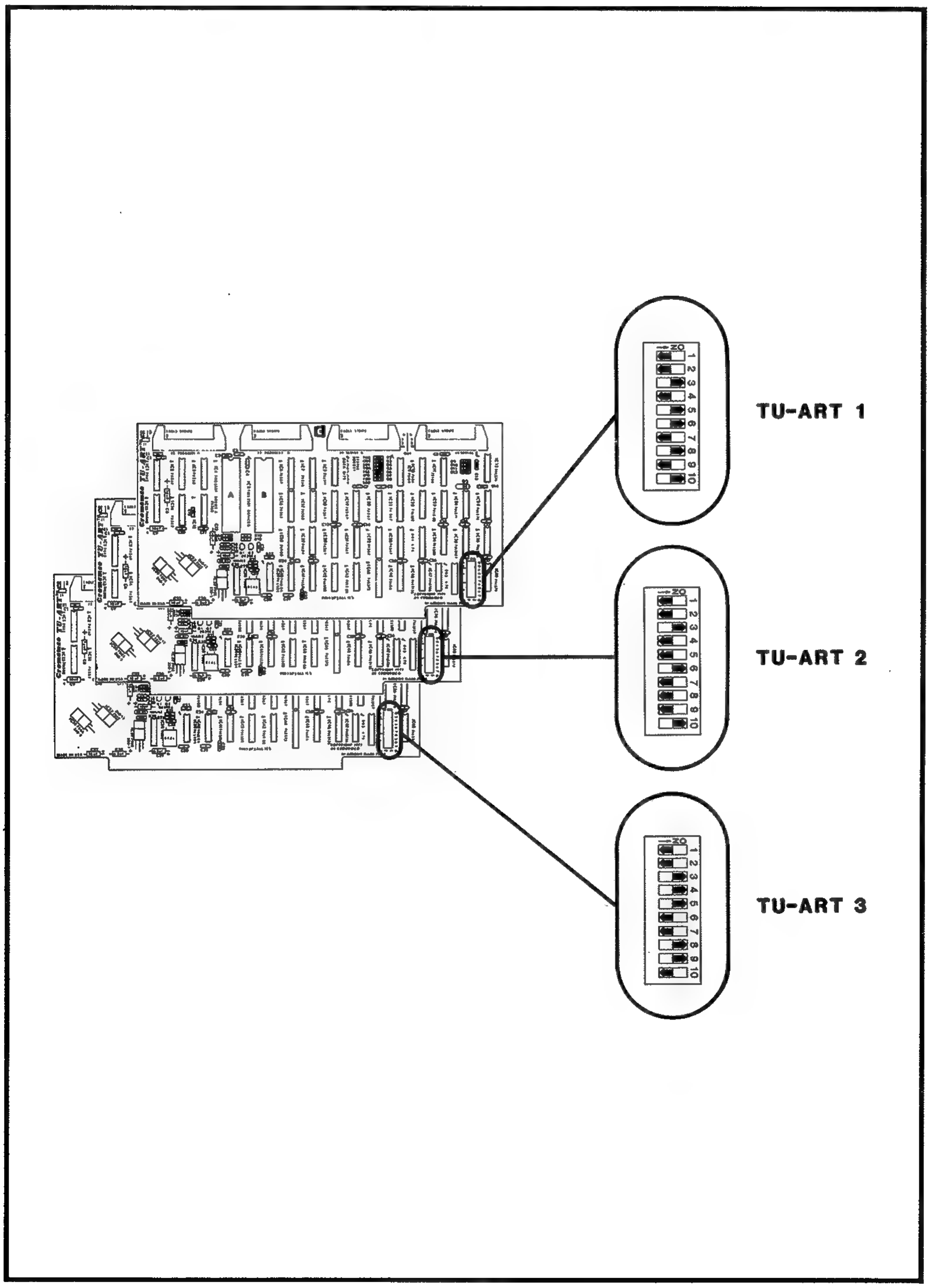
Real Time Clock

The Cromix Operating System recognizes one of two interrupt sources on the 16FDC Floppy Disk Controller as a real time clock generator. These are the 5501 Timer 3 and the jumper selectable 512 millisecond real time clock interrupt. The 512 millisecond real time clock interrupt is the more accurate of the two and should be used if a 16FDC is present in the system. The following modifications enable the Cromix Operating System to use the 512 millisecond clock. This modification is now made at the factory and, unless you have an older 16FDC board (prior to rev F1 mod level 3), it should not be necessary for you to make these changes. If the modifications are not implemented, the operating system defaults to 16FDC 5501 Timer 3. This option is only available using a 16FDC and a Cromix Operating System version 11.00 or greater.

TU-ART TERMINAL INTERFACE

The initial terminal (tty1) is connected to the port provided for this purpose on the floppy disk controller board. A minimum system does not require a TU-ART board.

Up to five additional terminals may be attached to the system by means of Cromemco TU-ART interface boards. Two terminals may be connected to each TU-ART so that a maximum of three of these boards will be needed.



TU-ART Switch Settings

TU-ART #1 services user(s) two and (if required) three through its serial ports A (port 20h) and B (port 50h), respectively. TU-ART #1 can also service parallel printers lptl (port 54) and lpt2 (port 24).

If the system has more than three users, TU-ART #2 services user(s) four and (if required) five through its serial ports A (port 60h) and B (port 70h), respectively.

In a six user system, TU-ART #3 services user six through its serial port A (port 80h).

Please refer to Chapter 6 and Appendix B if it is necessary to set up a Cromix system with more than six users.

PRINTER INTERFACE

The Cromemco PRI printer interface board supports one fully formed character printer such as the Cromemco 3355B and one dot matrix printer such as the Cromemco 3703.

All switches on the PRI should be OFF.

MINIMUM BOARD REVISION LEVELS

The following revision levels are the minimum for each board to run in a system controlled by the Cromix Operating System.

ZPU	Any revision level
64KZ	Rev. J or higher
TU-ART	Rev. E or higher
PRI	Rev. E or higher
4FDC	Rev. C or higher
16FDC	Rev. E or higher
IOP	Rev. C or higher
Quadart	Rev. C or higher

PRIORITY INTERRUPT CABLE

The priority interrupt cable is a single wire with several connectors at regular intervals along its length. This cable must run between all of the following boards in the system: 4FDC or 16FDC, all IOPs, TU-ARTs, and the PRI. If the system has no PRI, IOPs or TU-ARTs, then no priority interrupt cable is required.

THE PRIORITY INTERRUPT CABLE MUST NOT BE CONNECTED TO THE WDI, WDI-II OR ANY QUADARTS.

The cable must go from the priority interrupt cable connector **out** pin on the disk controller board to the **in** pin on the next board in sequence, and so on. Note that the positions of the in and out pins on the 16FDC board are reversed from the in and out pin positions on the other boards. The Priority Interrupt cable should run from the 16FDC (or 4FDC) to the TU-ART(s) to the IOP(s) to the PRI.

Appendix B

CONNECTING TERMINALS WITH THE IOP/QUADART

Terminals may be connected to a Cromemco computer running under the Cromix Operating System by using combinations of TU-ART and/or IOP/Quadart boards. This section covers hardware installation of the IOP and Quadart boards.

Background

The IOP/Quadart board combination reduces the overhead associated with character processing by utilizing distributed processing techniques. This reduces the burden on the central processing unit which in turn increases processor throughput.

Hardware Setup

The Cromix Operating System accommodates up to a total of four IOPs. Each IOP accommodates up to four Quadarts. This allows a theoretical total of 64 terminals. Note that the total number of I/O boards required to support 64 terminals is 20, which leaves only 1 slot in a 21 slot card cage for memory, disk controllers, etc. Cromix looks for a maximum of 18 terminals to be logged in, and the maximum practical number of terminals is 12. The number of user banks is still limited to 6. By allowing room in the software for 64 possible terminals, any one of the four possible IOPs may be installed in the system without major software modification.

If, for example, an IOP/Quadart board set is installed for use with the Cromemco RBTE software package at the IOP address CEh, an IOP/Quadart set could be installed at IOP address BEh (IOP-2) to support terminals qtty17 through qtty32, the number of terminals being dependent upon the number of Quadarts used.

To simplify installation and reference, each IOP and Quadart has been assigned a number in this manual (IOP 1 through 4 and Quadart 1 through 16). These numbers are used to refer to the IOPs and Quadarts corresponding to each qtty terminal.

IOP Switch Settings

Switch 1 of the IOP is the address selection switch. It should be set to address the IOPs as follows:

IOP Number	Base Address	Terminals Supported
IOP(1)	CEh	qtty1 - qtty16
IOP(2)	BEh	qtty17 - qtty32
IOP(3)	AEh	qtty33 - qtty48
IOP(4)	9Eh	qtty49 - qtty64

Refer to the switch settings at the end of this section and to Appendix D for device numbers.

IOP Priority

Each IOP must be connected in the S-100 priority interrupt chain. It is suggested that the IOP(s) should be connected after the 16FDC/4FDC and before the PRI. The 16FDC or 4FDC priority out is connected to priority in on the IOP, and the IOP priority out is connected to priority in on the PRI.

Quadart Switch Settings

Switch 1 of the Quadart is the address selection switch. It should be set to address the Quadarts as follows:

Quadart Number	Port Address
Quadart 1,5,9,13	40h
Quadart 2,6,10,14	60h
Quadart 3,7,11,15	80h
Quadart 4,8,12,16	A0h

Refer to the switch settings at the end of this section and to Appendix D for device numbers.

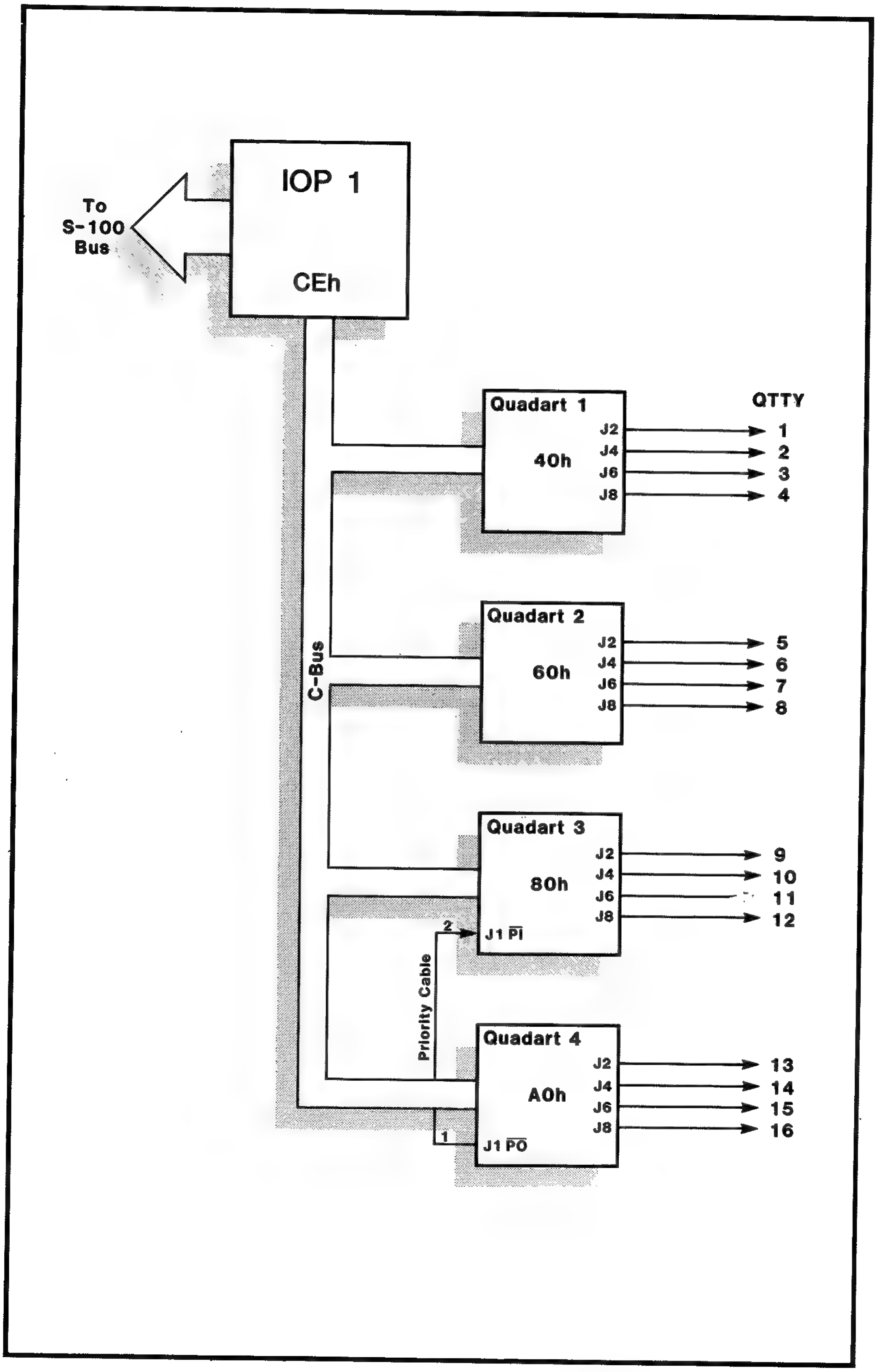
Quadart Priority

When Quadarts are used to support serial communications under the Cromix Operating System, Quadart priority must be established. Each of the four possible Quadarts that can be attached to each IOP is configured in the same manner. That is, the first three Quadarts attached to each IOP are prioritized via the priority header (J28) on each Quadart (IC 28 on the printed circuit board legend) and the fourth Quadart priority is established by connecting a priority cable from J1 pin 1 (priority out) on the third Quadart to J1 pin 2 (priority in) on the fourth Quadart. The priority header must be removed from the fourth Quadart (Quadart 4,8,12,16) in each IOP/Quadart board set.

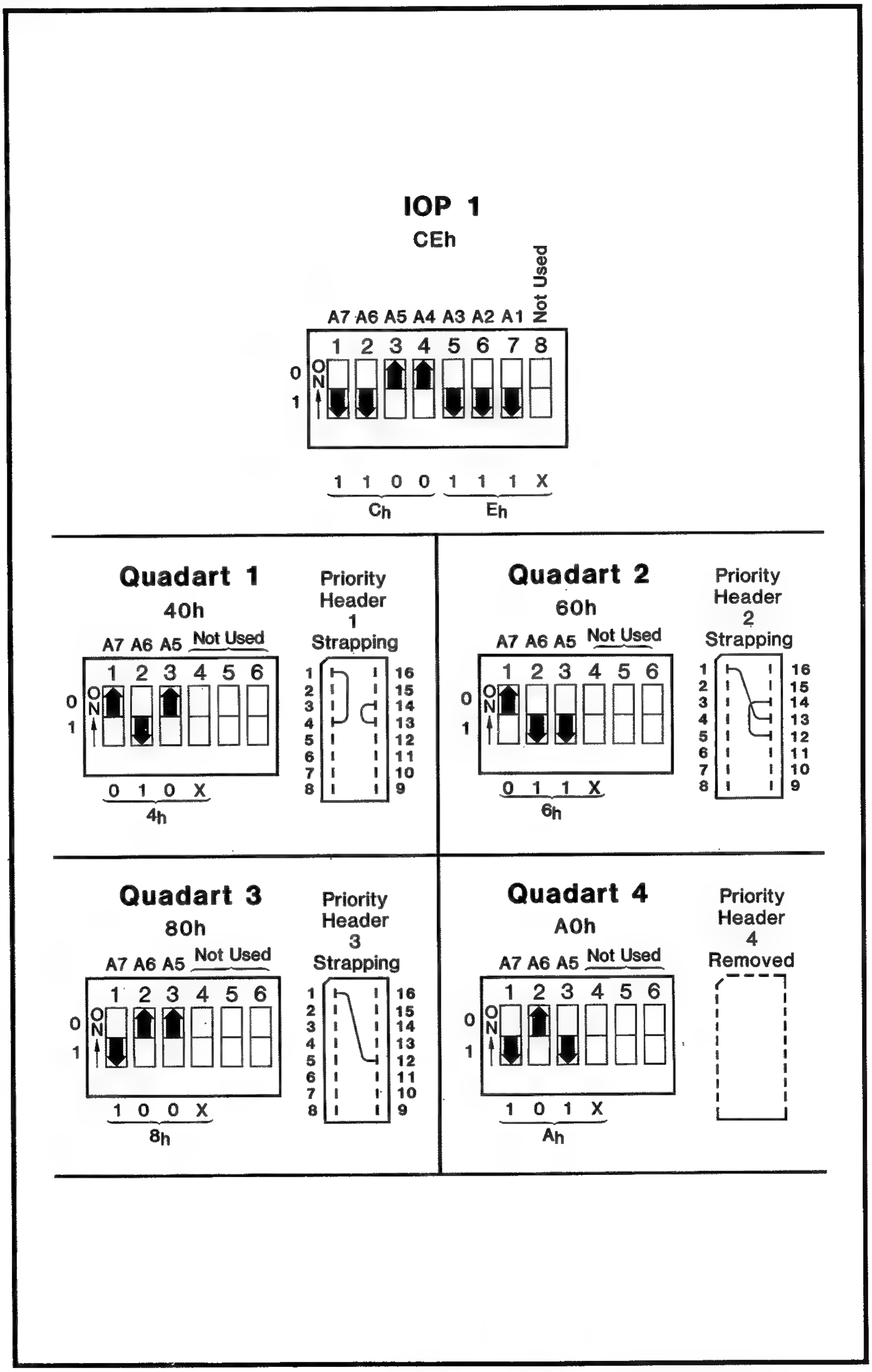
Do not connect the Quadart priority interrupt cable to any of the boards on the S-100 priority interrupt chain (16FDC, TU-ART, PRI, and so on), or vice-versa.

Terminal Connection

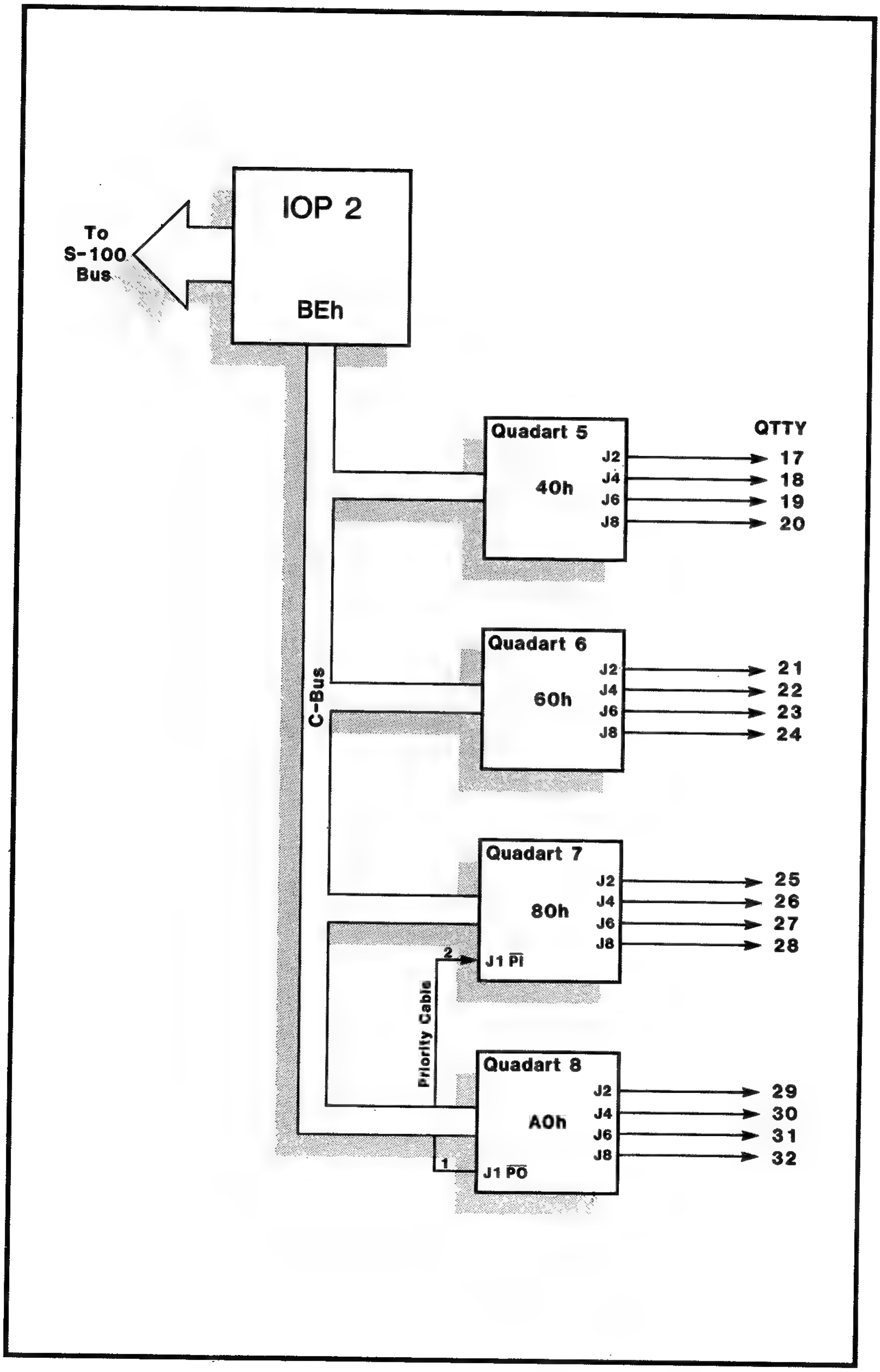
A 25-conductor ribbon cable must be installed in the computer for each terminal used. The end of the ribbon cable with the DB-25 connector should be attached to one of the DB-25 knockouts on the back panel of the computer. The ribbon cable is then routed to the Quadart and connected to one of the four 26 pin terminal connectors (J2, J4, J6, or J8). The 3-conductor cable supplied with the terminal is connected from the terminal to the DB-25 connector on the back panel of the computer.



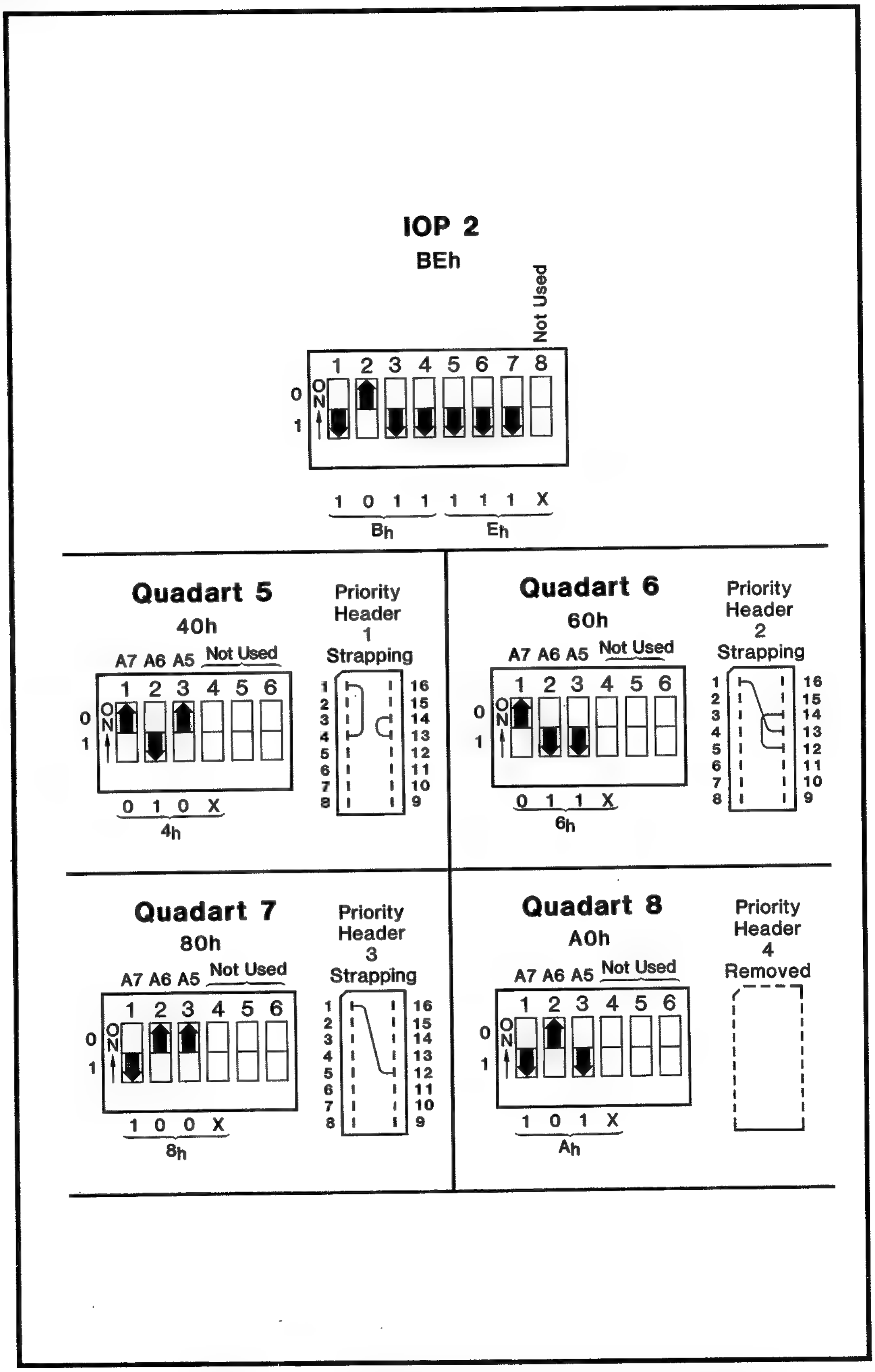
IOP 1/QUADART 1-4 BLOCK DIAGRAM



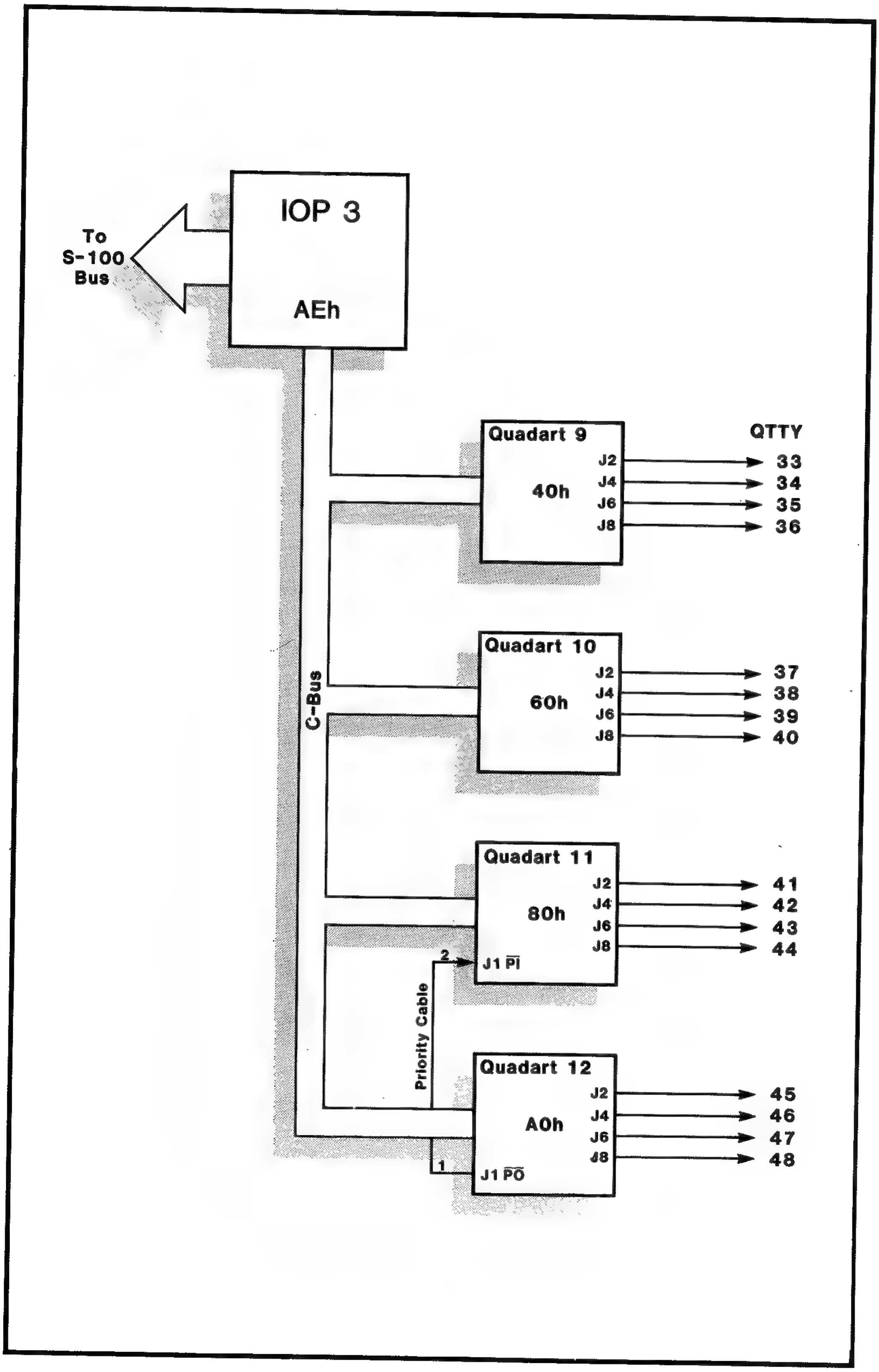
IOP 1/QUADART 1-4 SWITCH SETTING AND HEADER STRAPPING



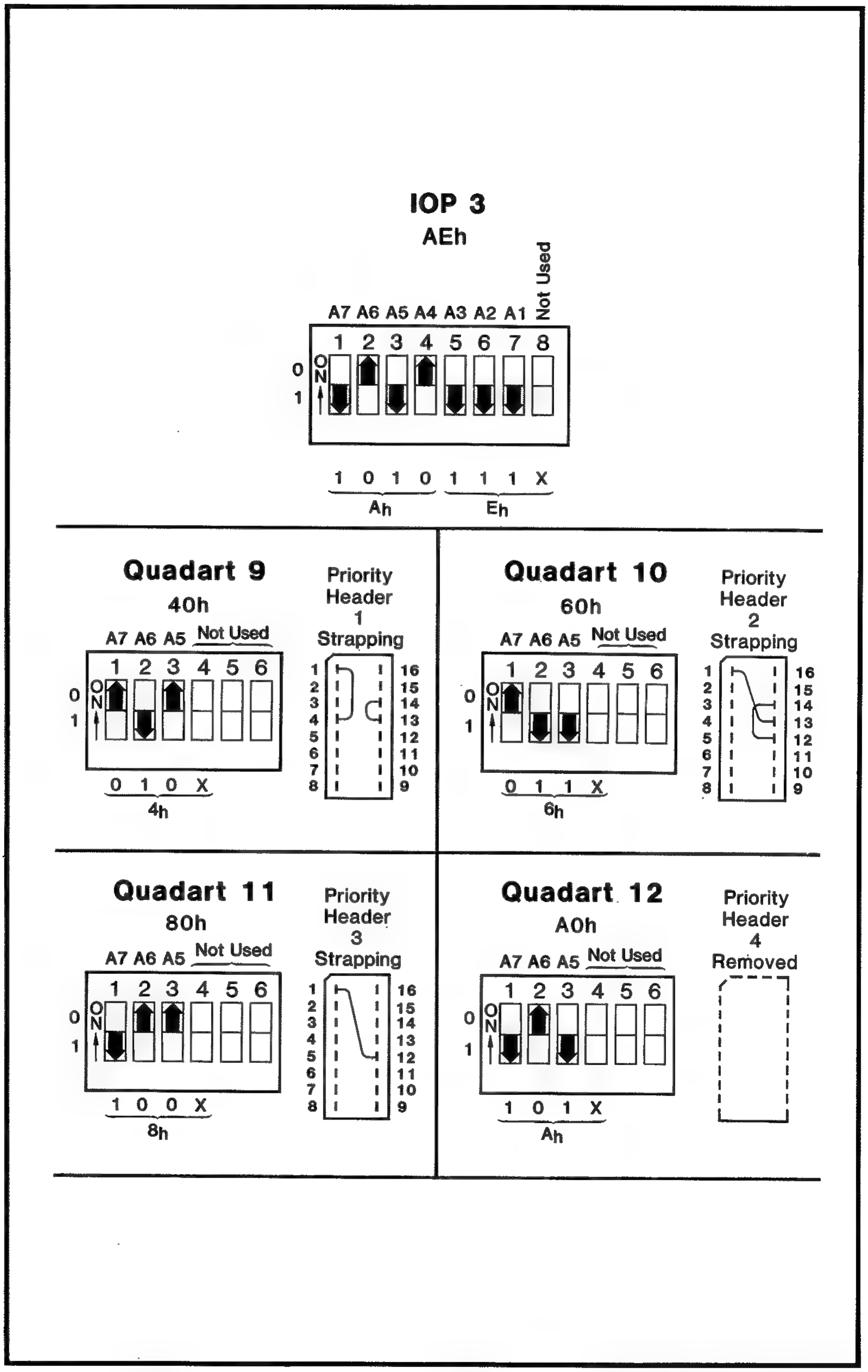
IOP 2/QUADART 5-8 BLOCK DIAGRAM



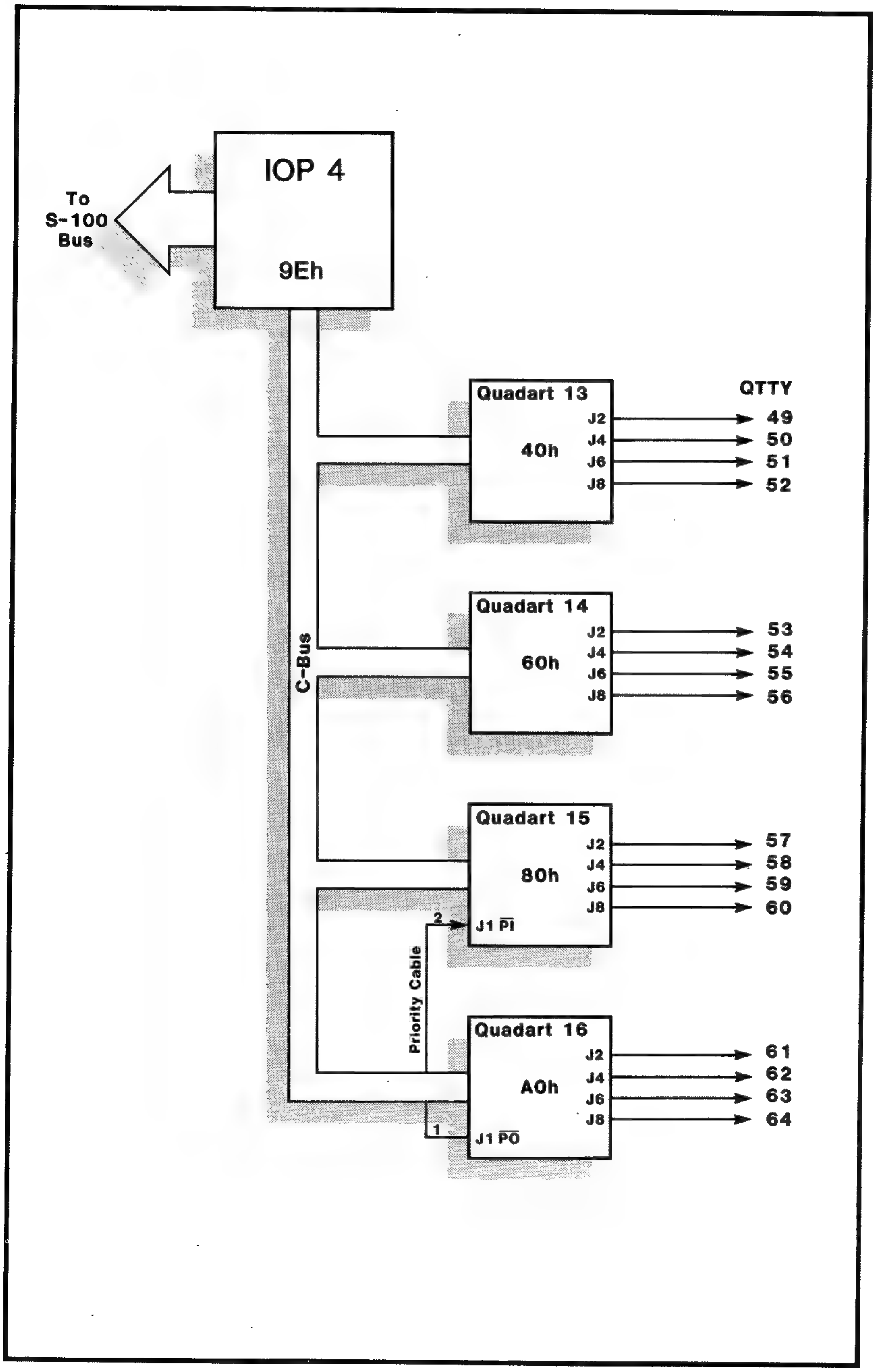
IOP 2/QUADART 5-8 SWITCH SETTING AND HEADER STRAPPING



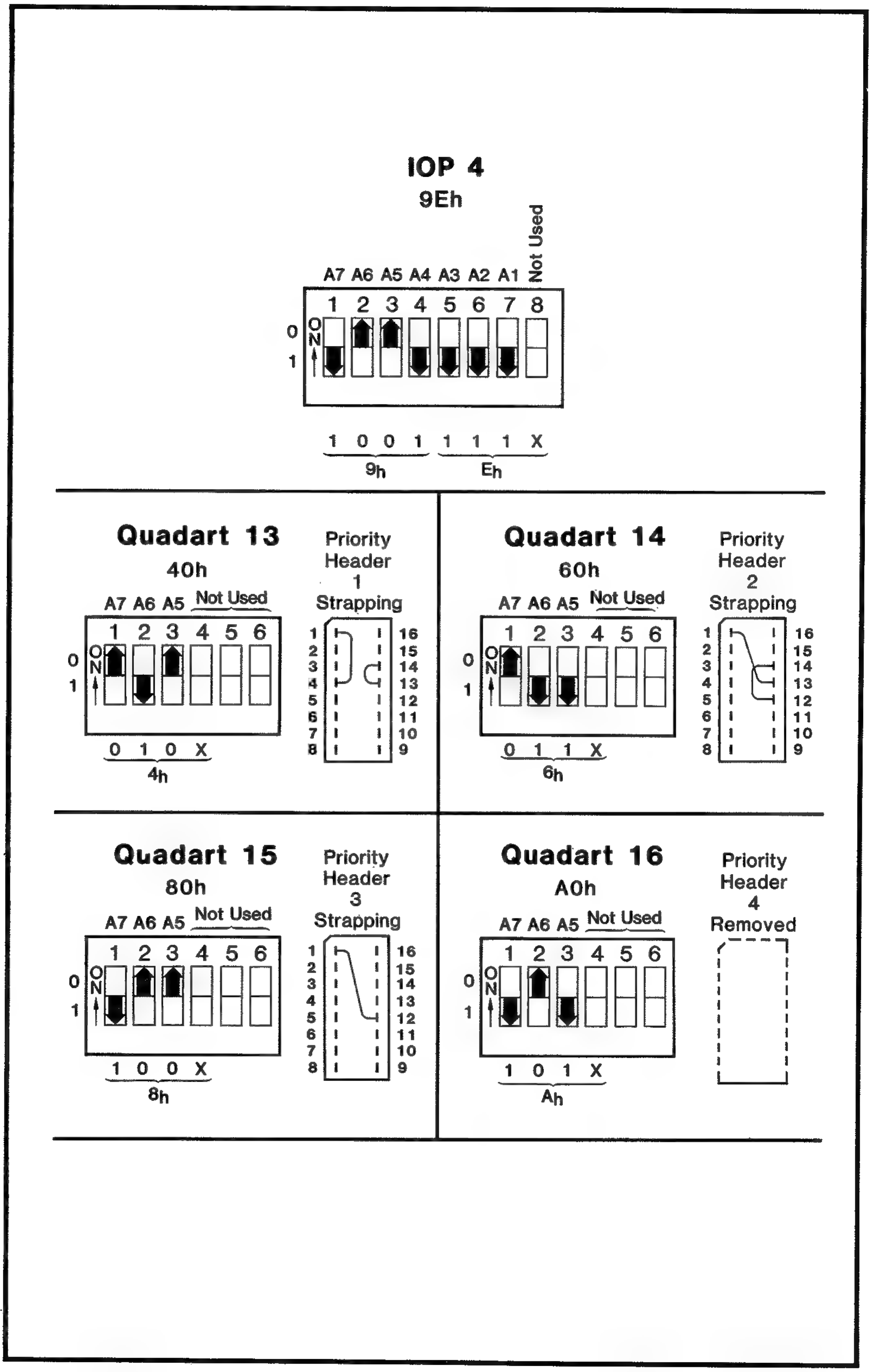
IOP 3/QUADART 9-12 BLOCK DIAGRAM



IOP 3/QUADART 9-12 SWITCH SETTING AND HEADER STRAPPING



IOP 4/QUADART 13-16 BLOCK DIAGRAM



IOP 4/QUADART 13-16 SWITCH SETTING AND HEADER STRAPPING

Appendix C

PROGRAMMER'S EQUATE LISTINGS

FILE NAME: JSYSEQU.Z80

```
stdin       equ  0      ;standard input channel
stdout      equ  1      ;standard output channel
stderr      equ  2      ;standard error channel

argc        equ  40H    ;location for argument count
argv        equ  42H    ;location for argument list vector
arg0        equ  0      ;arg offset
arg1        equ  2      ;arg offset
arg2        equ  4      ;arg offset
arg3        equ  6      ;arg offset
arg4        equ  8      ;arg offset
```

C-register modes for .create, .open

```
op.read     equ  0      ;read only
op.write    equ  1      ;write only
op.rdwr     equ  2      ;read and write
op.append   equ  3      ;append only
op.xread    equ  4      ;exclusive read only
op.xwrite   equ  5      ;exclusive write only
op.xrdwr    equ  6      ;exclusive read and write
op.xappend  equ  7      ;exclusive append only

op.truncf   equ  80H    ;truncate on create flag
op.condf    equ  40H    ;conditional create flag
```

C-register file position modes for .setpos

```
fwd.begin   equ   0     ;forward from the beginning of the file
fwd.current equ   1     ;forward from the current file position
fwd.end     equ   2     ;forward from the end of the file
bak.current equ  -1     ;backward from the current file position
bak.end     equ  -2     ;backward from the end of the file
```

C-register modes for .fstat, .cstat, .fchstat, .cchstat

```
st.all       equ  0      ;all of inode (128 bytes)
st.owner     equ  1      ;de = owner
st.group     equ  2      ;de = group
st.aowner    equ  3      ;d = owner access, e = mask
st.agroup    equ  4      ;d = group access, e = mask
st.aother    equ  5      ;d = other access, e = mask
st.ftype     equ  6      ;d = file type, e = special device #
st.size      equ  7      ;dehl = file size
st.nlinks    equ  8      ;de = number of links
st.inum      equ  9      ;de = inode number
st.device    equ  10     ;d = device number of file system containing inode
st.tcreate   equ  11     ;de-> time created
st.tmodify   equ  12     ;de-> time last modified
st.taccess   equ  13     ;de-> time last accessed
st.tdumped   equ  14     ;de-> time last dumped
st.devno     equ  15     ;de = device number if inode is a device
```

file types for st.ftype

```
is.ordin        defl    0       ;ordinary file
is.direct       defl    1       ;directory file
is.char         defl    2       ;character device
is.block        defl    3       ;block device
is.pipe         defl    4       ;pipe file
```

access bits for access flags

```
ac.read         defl    0       ;read access bit
ac.exec         defl    1       ;execute access bit
ac.writ         defl    2       ;write access bit
ac.apnd         defl    3       ;append access bit
```

C-register modes for .setuser, .getuser, .setgroup, .getgroup

```
id.effective    equ     0       ;effective id
id.login        equ     1       ;login id
id.program      equ     2       ;program id
id.hl           equ     3       ;id contained in HL register
```

Signals (applies to .kill and .signal)

```
sigabort        defv    1       ;CNTRL-C key
siguser         defv    2       ;user-specifiable key
sigkill         defv    3       ;kill (not catchable)
sigterm         defv    4       ;terminate (all but kill are catchable)
sigalarm        defv    5       ;alarm clock
sigpipe         defv    6       ;broken pipe
sighngup        defv    7       ;modem hang up

sigmax          defv    8       ;maximum signal number
```

System Call Numbers

```
.makdev     equ   00H     ;makdev(d,e,hl)         make device entry
.makdir     equ   01H     ;makdir(hl)             make a directory
.getdir     equ   02H     ;getdir(hl)             get current directory name
.setdir     equ   03H     ;setdir(hl)             change current directory

.mount      equ   04H     ;mount(c,de,hl)         mount file system
.unmount    equ   05H     ;unmount(hl)            unmount file system
.delete     equ   06H     ;delete(hl)             delete file
.chkdev     equ   07H     ;chkdev(d,e)            check for device driver

.create     equ   08H     ;b=create(c,hl)         create & open file
.open       equ   09H     ;b=open(c,hl)           open file
.chdup      equ   0AH     ;c=chdup(b)             duplicate channel
.close      equ   0BH     ;close(b)               close file

.exchg      equ   0CH     ;exchg(b,c)             exchange data in files
.trunc      equ   0DH     ;trunc(b)               truncate open file
.pipe       equ   0EH     ;b,c=pipe()             create a pipe
;           equ   0FH

.getpos     equ   10H     ;dehl=getpos(b)         get file position
.setpos     equ   11H     ;setpos(c,dehl)         set file position
.getmode    equ   12H     ;d=getmode(b,c)         get device characteristics
.setmode    equ   13H     ;d=setmode(b,c,d,e)     set device characteristics

.rdseq      equ   14H     ;de=rdseq(b,de,hl)      read n bytes
.wrseq      equ   15H     ;de=wrseq(b,de,hl)      write n bytes
.rdbyte     equ   16H     ;a=rdbyte(b)            read 1 byte
.wrbyte     equ   17H     ;wrbyte(b,a)            write 1 byte
.rdline     equ   18H     ;de=rdline(b,de,hl)     read a line
.wrline     equ   19H     ;de=wrline(b,hl)        write a line
;           equ   1AH
.printf     equ   1BH     ;printf(b,hl)           print formatted string
.error      equ   1CH     ;error(a,b,de,hl)       print error message

.fstat      equ   20H     ;fstat(c,de,hl)         get file status (inode)
.cstat      equ   21H     ;cstat(b,c,de)          get channel status (inode)
.fchstat    equ   22H     ;fchstat(c,de,hl)       change file status
```

```
.cchstat    equ  23H   ;cchstat(b,c,de)        change channel status
.flink      equ  24H   ;flink(de,hl)           link to file
.clink      equ  25H   ;clink(b,de)            link to open channel
.faccess    equ  26H   ;faccess(c,hl)          test file access
.caccess    equ  27H   ;caccess(b,c)           test channel access
;           equ  28H
;           equ  29H

.getdate    equ  30H   ;d,e,h,l=getdate()      get date
.setdate    equ  31H   ;setdate(e,h,l)         set date
.gettime    equ  32H   ;e,h,l=gettime()        get time
.settime    equ  33H   ;settime(e,h,l)         set time

.getuser    equ  34H   ;de,hl=getuser()        get user id
.setuser    equ  35H   ;setuser(hl)            set user id
.getgroup   equ  36H   ;de,hl=getgroup()       get group id
.setgroup   equ  37H   ;setgroup(hl)           set group id

.getprior   equ  38H   ;hl=getprior()          get process priority
.setprior   equ  39H   ;setprior(hl)           set process priority
.getproc    equ  3AH   ;hl=getproc()           get process id
;           equ  3BH
;           equ  3CH
;           equ  3DH
.lock       equ  3EH   ;lock(c,de,hl)          lock key
.unlock     equ  3FH   ;unlock(c,de,hl)        unlock key

.signal     equ  40H   ;signal(c,hl)           set up to receive a signal
.kill       equ  41H   ;kill(c,hl)             send a signal
.sleep      equ  42H   ;sleep(hl)              sleep for hl seconds
.alarm      equ  43H   ;alarm(hl)              set alarm clock
.pause      equ  44H   ;pause()                pause for alarm clock
.wait       equ  45H   ;c,de,hl=wait()         wait for child process
.exit       equ  46H   ;exit(hl)               exit process (close files)
;           equ  47H

.fshell     equ  48H   ;fshell(de)             fork a shell process
.shell      equ  49H   ;shell(de)              transfer to shell process
;           equ  4AH
.fexec      equ  4BH   ;fexec(bc,de,hl)        fork and execute program
.exec       equ  4CH   ;exec(bc,de,hl)         execute program
;           equ  4DH
;           equ  4EH
;           equ  4FH

;           equ  50H
.indirect   equ  51H   ;indirect(a,b,c,de,hl)  system call in A-register

.update     equ  52H   ;update()               update disk I/O buffers
.mult       equ  53H   ;dehl=mult(bc,hl)       multiply
.divd       equ  54H   ;de,hl=divd(dehl,bc)    divide
.version    equ  55H   ;hl=version()           get system version #
;           equ  56H
```

error code definitions

```
?badchan        defv    1       ;bad channel #
?toomany        defv    2       ;channel already open
?notopen        defv    3       ;channel not open
?endfile        defv    4       ;end-of-file
?ioerror        defv    5       ;I/O error
?filtable       defv    6       ;file table exhausted
?notexist       defv    7       ;file does not exist
?badname        defv    8       ;bad file name
?diraccess      defv    9       ;directory access
?filaccess      defv    10      ;file access
?exists         defv    11      ;file already exists
?nospace        defv    12      ;no disk space left
?noinode        defv    13      ;no inodes left
?inotable       defv    14      ;inode table exhausted
?badcall        defv    15      ;illegal system call
?filsize        defv    16      ;file size too big
?mnttable       defv    17      ;mount table exhausted
?notdir         defv    18      ;not a directory
?isdir          defv    19      ;is a directory
?priv           defv    20      ;privileged system call
?notblk         defv    21      ;not a block special device
?fsbusy         defv    22      ;file system busy
?notordin       defv    23      ;not an ordinary file
?notmount       defv    24      ;device not mounted
?nochild        defv    25      ;no child processes
?nomemory       defv    26      ;not enough memory
?ovflo          defv    27      ;divide overflow
?argtable       defv    28      ;argument table exhausted
?arglist        defv    29      ;bad argument list
?numlinks       defv    30      ;too many number of links
?difdev         defv    31      ;cross-device link
?nodevice       defv    32      ;no special device
?usrtable       defv    33      ;user process table exhausted
?badvalue       defv    34      ;value out of range
?notconn        defv    35      ;I/O device not connected
?devopen        defv    36      ;device open error
?diruse         defv    37      ;directory in use (delete)
?filuse         defv    38      ;file in use (exclusive access)
?nomatch        defv    39      ;no match on ambiguous name
?chnaccess      defv    40      ;channel access
?notcromix      defv    41      ;not a cromix disk
?badfree        defv    42      ;bad free list
?badinum        defv    43      ;bad inode number
?readonly       defv    44      ;device mounted for read only
?noproc         defv    45      ;process does not exist
?signal         defv    46      ;system call was aborted
?badpipe        defv    47      ;bad call on a pipe
?locked         defv    48      ;locked
?deadlock       defv    49      ;deadlocked
?lcktable       defv    50      ;lock table exhausted
```

FILE NAME: /EQU/TMODEEQU.Z80

```
list    off
list    noxref     ; (use this line only with ASMB version 3.08 or later)


; Mode definitions for TP tape devices and
; c-register values for .GETMODE and .SETMODE system calls

tpmmin          equu    -60             ; minimum mode number
TPABORT         equ     tpmmin + 0      ; re-initialize tape driver
TPFMARK         equ     tpmmin + 2      ; write file mark
TPSECURE        equ     tpmmin + 3      ; security erase
TPREWIND        equ     tpmmin + 4      ; rewind
TPUNLOAD        equ     tpmmin + 5      ; rewind and unload
TPMODE          equ     tpmmin + 6      ; mode bits
TPFILNO         equ     tpmmin + 7      ; file number
TPBLKNO         equ     tpmmin + 8      ; block number
TPOBLKLN        equ     tpmmin + 9      ; block length for next block written
TPIBLKLN        equ     tpmmin + 10     ; block length of last block read
TPOBLKS         equ     tpmmin + 11     ; number of blocks written
TPSTAT          equ     tpmmin + 12     ; get error (status-2, status-1)

tpgmmin         equ     TPMODE          ; minimum getmode number
tpgmmax         equ     TPSTAT          ; maximum getmode number

tpsmmin         equ     TPABORT         ; minimum setmode number
tpsmmax         equ     TPOBLKLN        ; maximum setmode number

; TPMODE bits
EOFCLOSE        equ     7               ; write EOF to tape when device closes

; TPSTAT status bits, returned in e-register (obtained from PIO input port A)
DRVBUSY         equ     7               ; drive busy
WRRDY           equ     6               ; FIFO ready for input (used for write)
RDRDY           equ     5               ; FIFO output ready (used for read)
LOADPT          equ     4               ; load point
FBUSY           equ     3               ; formatter busy
ONLINE          equ     2               ; on line
IDENT           equ     1               ; ident?
RDY             equ     0               ; ready
```

```
; TPSTAT status bits, returned in d-register (obtained from PIO input port B)
HISPEED         equu     7               ; high speed status
HARDERR         equ      5               ; hard error
FLMARK          equ      4               ; file mark
CORERR          equ      3               ; correctable error
WRPROT          equ      2               ; file write-protected
EOT             equ      1               ; end of tape
RWINDING        equ      0               ; rewinding

        list    xref    ; (use this line only with ASMB version 3.08 or later)
        list    on
```

FILE NAME: /EQU/MODEEQU.Z80

```
    list off
    list noxref  ; (use this line only with ASMB version 3.08 or later)

; ---------------------------------------------------------------------
; mode definitions for terminals and printers,
; TTY, QTTY, MTTY, LPT, SLPT, QSLPT, and TYP

; c-register values for .GETMODE and .SETMODE system calls
MD_ISPEED       defl    0               ; input speed
MD_OSPEED       defl    1               ; output speed
MD_MODE1        defl    2               ; flags: RAW, ECHO, etc.
MD_MODED        defl    3               ; delays for NL, CR, etc.
MD_MODE2        defl    4               ; flags: PAUSE, XFF, etc.
MD_MODE3        defl    5               ; flags: CBREAK, VRAW, etc.
MD_ERASE        defl    6               ; auxiliary erase character
MD_DELECHO      defl    7               ; erasure echo character
MD_LKILL        defl    8               ; line kill character
MD_USIGNAL      defl    9               ; SIGUSER signal key
MD_LENGTH       defl    10              ; page length (lines)
MD_WIDTH        defl    11              ; page width (columns)
MD_BMARGIN      defl    12              ; bottom margin (lines)
MODELEN equ     MD_BMARGIN + 1
; more c-register values for .GETMODE and .SETMODE system calls.
MD_STATUS       equ     -100            ; check whether input queues empty
MD_IFLUSH       equ     -101            ; flush input queues
MD_FNKEYS       equ     -104            ; turn function keys on or off
MD_PSIGHUP      equ     -105            ; signal current process if hang up
;               equ     -106            ; (this value reserved)


; d-register values for MD_ISPEED baudrate calls
S_HANGUP        equ     0               ; hang up phone
;               equ     1               ; 50 baud
;               equ     2               ; 75 baud
S_110           equ     3               ; 110 baud
;               equ     4               ; 134.5 baud
S_150           equ     5               ; 150 baud
;               equ     6               ; 200 baud
S_300           equ     7               ; 300 baud
;               equ     8               ; 600 baud
S_1200          equ     9               ; 1200 baud
;               equ     10              ; 1800 baud
S_2400          equ     11              ; 2400 baud
S_4800          equ     12              ; 4800 baud
S_9600          equ     13              ; 9600 baud
;               equ     14              ; External A
;               equ     15              ; External B
S_19200         equ     16              ; 19200 baud

S_CTSWAIT       equ     125     ; wait for Clear To Send
S_NOCHG         equ     126     ; no change of baudrate
```

```
S_UNINIT        equ     127     ; uninitialized baudrate
Sfl_AUTO        equ     7       ; (bit 7) input CRs from keyboard to set baudr

; d-register & e-register bits for MD_MODE1 calls
TANDEM          defl    0       ; send XOFF/XON to control filling of input buf
XTAB            defl    1       ; expand TABs
LCASE           defl    2       ; convert alphabetics to lower case
ECHO            defl    3       ; echo input
CRDEVICE        defl    4       ; on input, map CR into NL,
                                ; on output, change NL to CRLF.
RAW             defl    5       ; on input, return after each character,
                                ; no erase, linekill, or EOF characters,
                                ; no output PAUSE or output width truncation,
                                ; treat XOFF/XON as regular input.
ODD             defl    6       ; parity function bits
EVEN            defl    7       ;

; d-register & e-register values for MD_MODED calls
NLDELAY         defl    03H     ; (pairs of bits)
TABDELAY        defl    0CH     ;
CRDELAY         defl    30H     ;
FFDELAY         defl    40H     ; (single bits)
BSDELAY         defl    80H     ;

; d-register & e-register bits for MD_MODE2 calls
PAUSE           defl    0       ; wait for CNTRL-Q after a page is output
NOTIMMECHO      defl    1       ; do not echo characters typed-ahead
NOECNL          defl    2       ; do not echo NLs
SGENABLE        defl    3       ; send SIGUSER signal if MD_USIGNAL key pushed
ABENABLE        defl    4       ; send SIGABORT signal if CNTRL-C key pushed
XFF             defl    5       ; expand FFs
WRAP            defl    6       ; wrap-around if page width is exceeded
SIGALLC         defl    7       ; send SIGUSER signal for every key pushed

; d-register & e-register bits for MD_MODE3 calls
ESCRETN         defl    0       ; ESC causes input line to be returned
FNKEYS          defl    1       ; response to 3102 function keys enabled
HUPENAB         defl    2       ; hang up modem when device is finally closed
SIGHUPALL       defl    3       ; send SIGHANGUP signals to all processes which
                                ; use this TTY device if modem hangs up
CBREAK          defl    4       ; on input, return after each character,
                                ; no erase, linekill, or EOF characters.
BINARY          defl    5       ; on input, return after each character,
                                ; no erase, linekill, or EOF characters,
                                ; no output PAUSE or output width truncation,
                                ; treat XOFF/XON as regular input,
                                ; no tandem mode (i.e., no input buf control),
                                ; no abort signal (^C), no user signal,
                                ; no changing or checking parity bit,
                                ; no delays after control chars such as NLs,
                                ; no echoing,
                                ; no character transformations (i.e., ignore
                                ; the LCASE, CRDEV, and XTABS modes)
```

```
                                         ; no function-key decoding.
DISCARD         defl    7                ; discard the device when it is no longer open

; d-register bits for MD_STATUS calls
INOTEMPTY       defl    0                ; there is a character in the input buffer
                                         ; (but if not CBREAK, RAW, or BINARY mode,
                                         ; it won't be accessible until a whole line
                                         ; is entered)

; ---------------------------------------------------------------------------
; old names
md.ibaud        equu    md_ispeed
md.obaud        equ     md_ospeed
md.mode1        equ     md_mode1
md.mode2        equ     md_mode2
md.mode3        equ     md_mode3
md.erase        equ     md_erase
md.dlecho       equ     md_delecho
md.kill         equ     md_lkill
md.signal       equ     md_usignal
md.length       equ     md_length
md.width        equ     md_width
md.bmargin      equ     md_bmargin

b.9600  equ     S_9600
b.19200 equ     S_19200
b.auto  equ     ^Sf1_AUTO

md1.tab         defl    XTAB
md1.echo        defl    ECHO
md1.cr.nl       defl    CRDEVICE
md1.raw         defl    RAW
md1.odd         defl    ODD
md1.even        defl    EVEN

md2.pause       defl    PAUSE
md2.later       defl    NOTIMMECHO
md2.noecnl      defl    NOECNL
md2.sgenable    defl    SGENABLE
md2.abenable    defl    ABENABLE
md2.ff          defl    XFF
md2.wrap        defl    WRAP

md2.esccr       defl    ESCRETN

st.charrdy      equ     INOTEMPTY

hangup          equ     HUPENAB
huptty          equ     SIGHUPALL

        list    xref    ; (use this line only with ASMB version 3.08 or later)
        list    on
```

SUPER BLOCK DEFINITIONS

```
frbcount     equu    80                ;free block list size
fricount     equ     80                ;free inode list size
frbsize      equ     frbcount *4+2     ;free list size in bytes
frisize      equ     fricount *2+2     ;free list size in bytes
             struct  0
sb.version   defs    2                 ;version number
sb.cromix    defs    6                 ;'cromix'
sb.istart    defs    2                 ;first inode block
sb.isize     defs    2                 ;number of inodes
sb.fsize     defs    4                 ;max block number
sb.time      defs    6                 ;last modified time
             mend    struct
             struct  512-frbsize-frisize
sb.nfree     defs    2                 ;free block count
sb.free      defs    frbcount*4        ;free list address
sb.ilist     defs    0                 ;i-list address
sb.ninode    defs    2                 ;free inode count
sb.inode     defs    fricount*2        ;free inodes
             mend    struct
```

INODE BUFFER DEFINITIONS

```
             struct  0
             defs    2                 ;avail list pointers
             defs    2
in.devn      defs    2                 ;inode device number
in.inum      defs    2                 ;inode number
in.flags     defs    1                 ;flags byte
in.ucount    defs    1                 ;usage count

in.begin     defs    0                 ;beginning of inode on disk
in.owner     defs    2                 ;file owner's user id
in.group     defs    2                 ;file owner's group id
in.aowner    defs    1                 ;owner access
in.agroup    defs    1                 ;group access
in.aother    defs    1                 ;other access
in.stat      defs    1                 ;file status
in.nlinks    defs    1                 ;number of links to inode
             defs    1
in.size      defs    4                 ;file total size (in bytes)
in.inode     defs    2                 ;this inode number
in.parent    defs    2                 ;parent inode number (for directories
                                        only)
in.sdevn     defs    0                 ;special device major & minor numbers
in.dcount    defs    2                 ;number entries in a directory
in.usage     defs    4                 ;number blocks actually used in file
in.tcreate   defs    6                 ;time created
in.tmodify   defs    6                 ;time last modified
in.taccess   defs    6                 ;time last accessed
in.tdumped   defs    6                 ;time last dumped (backed up)
```

```
in.index     defs    4*20          ;block pointers
inosize      defs    0             ;total inode size in bytes
             mend    struct
inocount     equ     20            ;size of inode table
is.type      defl    7             ;file type mask (in.stat)
is.ordin     defl    0             ;ordinary file
is.direct    defl    1             ;directory file
is.char      defl    2             ;character device
is.block     defl    3             ;block device
is.pipe      defl    4             ;pipe file
is.alloc     defl    7             ;inode allocated (bit in in.stat)
if.lock      defl    0             ;inode locked (in use by a process)
if.want      defl    1             ;inode wanted by another process
if.modf      defl    2             ;inode has to be written out
if.modt      defl    3             ;update time modified
if.acct      defl    4             ;update time accessed

ac.read      defl    0             ;read access bit
ac.exec      defl    1             ;execute access bit
ac.writ      defl    2             ;write access bit
ac.apnd      defl    3             ;append access bit
```

DIRECTORY FORMAT DEFINITIONS

```
             struct  0
dr.name      defs    24            ;name of entry
namsize      defs    0             ;size of name
             defs    4             ;reserved
dr.stat      defs    2             ;status & flags
dr.inum      defs    2             ;inode number of file
dirsize      defs    0             ;directory entry size (32 bytes)
             mend    struct

ds.alloc     equ     7             ;entry allocated bit
```

Appendix D

DEVICE DEFINITIONS

Device Definitions for ttys

Device Name	Board - Base port(hex)		Device number major : minor
tty1	16fdc	00h	1:0
tty2	TU-ART A(1)	20h	1:2
tty3	TU-ART B(1)	50h	1:5
tty4	TU-ART A(2)	60h	1:6
tty5	TU-ART B(2)	70h	1:7
tty6	TU-ART A(3)	80h	1:8
tty7	TU-ART B(3)	90h	1:9
tty8	TU-ART A(4)	A0h	1:10
tty9	TU-ART B(4)	B0h	1:11

Device Definitions for qttys

Device Name	IOP - Base Port		Quadart Base(hex)		Device number major : minor
qtty1	iop(1)	CEh	quadart(1)	40h	2:0
qtty2	iop(1)	CEh	quadart(1)	40h	2:1
qtty3	iop(1)	CEh	quadart(1)	40h	2:2
qtty4	iop(1)	CEh	quadart(1)	40h	2:3
qtty5	iop(1)	CEh	quadart(2)	60h	2:4
qtty6	iop(1)	CEh	quadart(2)	60h	2:5
qtty7	iop(1)	CEh	quadart(2)	60h	2:6
qtty8	iop(1)	CEh	quadart(2)	60h	2:7
qtty9	iop(1)	CEh	quadart(3)	80h	2:8
qtty10	iop(1)	CEh	quadart(3)	80h	2:9
qtty11	iop(1)	CEh	quadart(3)	80h	2:10
qtty12	iop(1)	CEh	quadart(3)	80h	2:11
qtty13	iop(1)	CEh	quadart(4)	A0h	2:12
qtty14	iop(1)	CEh	quadart(4)	A0h	2:13
qtty15	iop(1)	CEh	quadart(4)	A0h	2:14
qtty16	iop(1)	CEh	quadart(4)	A0h	2:15
qtty17	iop(2)	BEh	quadart(5)	40h	2:16
qtty18	iop(2)	BEh	quadart(5)	40h	2:17
qtty19	iop(2)	BEh	quadart(5)	40h	2:18
qtty20	iop(2)	BEh	quadart(5)	40h	2:19
qtty21	iop(2)	BEh	quadart(6)	60h	2:20
qtty22	iop(2)	BEh	quadart(6)	60h	2:21
qtty23	iop(2)	BEh	quadart(6)	60h	2:22
qtty24	iop(2)	BEh	quadart(6)	60h	2:23
qtty25	iop(2)	BEh	quadart(7)	80h	2:24
qtty26	iop(2)	BEh	quadart(7)	80h	2:25
qtty27	iop(2)	BEh	quadart(7)	80h	2:26
qtty28	iop(2)	BEh	quadart(7)	80h	2:27
qtty29	iop(2)	BEh	quadart(8)	A0h	2:28
qtty30	iop(2)	BEh	quadart(8)	A0h	2:29
qtty31	iop(2)	BEh	quadart(8)	A0h	2:30
qtty32	iop(2)	BEh	quadart(8)	A0h	2:31
qtty33	iop(3)	AEh	quadart(9)	40h	2:32
qtty34	iop(3)	AEh	quadart(9)	40h	2:33
qtty35	iop(3)	AEh	quadart(9)	40h	2:34
qtty36	iop(3)	AEh	quadart(9)	40h	2:35
qtty37	iop(3)	AEh	quadart(10)	60h	2:36
qtty38	iop(3)	AEh	quadart(10)	60h	2:37
qtty39	iop(3)	AEh	quadart(10)	60h	2:38
qtty40	iop(3)	AEh	quadart(10)	60h	2:39
qtty41	iop(3)	AEh	quadart(11)	80h	2:40
qtty42	iop(3)	AEh	quadart(11)	80h	2:41
qtty43	iop(3)	AEh	quadart(11)	80h	2:42
qtty44	iop(3)	AEh	quadart(11)	80h	2:43
qtty45	iop(3)	AEh	quadart(12)	A0h	2:44

Device Name	IOP - Base port		Quadart - base(hex)		Device number major : minor
qtty46	iop(3)	AEh	quadart(12)	A0h	2:45
qtty47	iop(3)	AEh	quadart(12)	A0h	2:46
qtty48	iop(3)	AEh	quadart(12)	A0h	2:47
qtty49	iop(4)	9Eh	quadart(13)	40h	2:48
qtty50	iop(4)	9Eh	quadart(13)	40h	2:49
qtty51	iop(4)	9Eh	quadart(13)	40h	2:50
qtty52	iop(4)	9Eh	quadart(13)	40h	2:51
qtty53	iop(4)	9Eh	quadart(14)	60h	2:52
qtty54	iop(4)	9Eh	quadart(14)	60h	2:53
qtty55	iop(4)	9Eh	quadart(14)	60h	2:54
qtty56	iop(4)	9Eh	quadart(14)	60h	2:55
qtty57	iop(4)	9Eh	quadart(15)	80h	2:56
qtty58	iop(4)	9Eh	quadart(15)	80h	2:57
qtty59	iop(4)	9Eh	quadart(15)	80h	2:58
qtty60	iop(4)	9Eh	quadart(15)	80h	2:59
qtty61	iop(4)	9Eh	quadart(16)	A0h	2:60
qtty62	iop(4)	9Eh	quadart(16)	A0h	2:61
qtty63	iop(4)	9Eh	quadart(16)	A0h	2:62
qtty64	iop(4)	9Eh	quadart(16)	A0h	2:63

Device Definition for lpts
(dot matrix printers)

Device Name	Board - Base port(hex)			Device number major : minor
lpt1	pri(1) or TU-ART(1)	B	50h	5:5
lpt2	pri(2) or TU-ART(1)	A	20h	5:2
lpt3	pri(3) or TU-ART(2)	A	60h	5:6
lpt4	pri(4) or TU-ART(2)	B	70h	5:7
lpt5	pri(5) or TU-ART(3)	A	80h	5:8
lpt6	pri(6) or TU-ART(3)	B	90h	5:9
lpt7	pri(7) or TU-ART(4)	A	A0h	5:10
lpt8	pri(8) or TU-ART(4)	B	B0h	5:11

Device Definitions for typs
(fully formed character printers)

Device Name	Board - Base port(hex)		Device number major : minor
typ1	pri(1)	50h	6:5
typ2	pri(2)	20h	6:2

Device Definition for qslpts (Quadart serial printers)

Device name	IOP - Base port		Quadart	base(hex)	Device number major : minor
qslpt1	iop(1)	CEh	quadart(1)	40h	9:0
qslpt2	iop(1)	CEh	quadart(1)	40h	9:1
qslpt3	iop(1)	CEh	quadart(1)	40h	9:2
qslpt4	iop(1)	CEh	quadart(1)	40h	9:3
qslpt5	iop(1)	CEh	quadart(2)	60h	9:4
qslpt6	iop(1)	CEh	quadart(2)	60h	9:5
qslpt7	iop(1)	CEh	quadart(2)	60h	9:6
qslpt8	iop(1)	CEh	quadart(2)	60h	9:7
qslpt9	iop(1)	CEh	quadart(3)	80h	9:8
qslpt10	iop(1)	CEh	quadart(3)	80h	9:9
qslpt11	iop(1)	CEh	quadart(3)	80h	9:10
qslpt12	iop(1)	CEh	quadart(3)	80h	9:11
qslpt13	iop(1)	CEh	quadart(4)	A0h	9:12
qslpt14	iop(1)	CEh	quadart(4)	A0h	9:13
qslpt15	iop(1)	CEh	quadart(4)	A0h	9:14
qslpt16	iop(1)	CEh	quadart(4)	A0h	9:15
qslpt17	iop(2)	BEh	quadart(5)	40h	9:16
qslpt18	iop(2)	BEh	quadart(5)	40h	9:17
qslpt19	iop(2)	BEh	quadart(5)	40h	9:18
qslpt20	iop(2)	BEh	quadart(5)	40h	9:19
qslpt21	iop(2)	BEh	quadart(6)	60h	9:20
qslpt22	iop(2)	BEh	quadart(6)	60h	9:21
qslpt23	iop(2)	BEh	quadart(6)	60h	9:22
qslpt24	iop(2)	BEh	quadart(6)	60h	9:23
qslpt25	iop(2)	BEh	quadart(7)	80h	9:24
qslpt26	iop(2)	BEh	quadart(7)	80h	9:25
qslpt27	iop(2)	BEh	quadart(7)	80h	9:26
qslpt28	iop(2)	BEh	quadart(7)	80h	9:27
qslpt29	iop(2)	BEh	quadart(8)	A0h	9:28
qslpt30	iop(2)	BEh	quadart(8)	A0h	9:29
qslpt31	iop(2)	BEh	quadart(8)	A0h	9:30
qslpt32	iop(2)	BEh	quadart(8)	A0h	9:31
qslpt33	iop(3)	AEh	quadart(9)	40h	9:32
qslpt34	iop(3)	AEh	quadart(9)	40h	9:33
qslpt35	iop(3)	AEh	quadart(9)	40h	9:34
qslpt36	iop(3)	AEh	quadart(9)	40h	9:35
qslpt37	iop(3)	AEh	quadart(10)	60h	9:36
qslpt38	iop(3)	AEh	quadart(10)	60h	9:37
qslpt39	iop(3)	AEh	quadart(10)	60h	9:38
qslpt40	iop(3)	AEh	quadart(10)	60h	9:39
qslpt41	iop(3)	AEh	quadart(11)	80h	9:40
qslpt48	iop(3)	AEh	quadart(12)	A0h	9:47

Device name	IOP - Base port(hex)		Quadart - base(hex)		Device number major : minor
qslpt42	iop(3)	AEh	quadart(11)	80h	9:41
qslpt43	iop(3)	AEh	quadart(11)	80h	9:42
qslpt44	iop(3)	AEh	quadart(11)	80h	9:43
qslpt45	iop(3)	AEh	quadart(12)	A0h	9:44
qslpt46	iop(3)	AEh	quadart(12)	A0h	9:45
qslpt47	iop(3)	AEh	quadart(12)	A0h	9:46
qslpt49	iop(4)	9Eh	quadart(13)	40h	9:48
qslpt50	iop(4)	9Eh	quadart(13)	40h	9:49
qslpt51	iop(4)	9Eh	quadart(13)	40h	9:50
qslpt52	iop(4)	9Eh	quadart(13)	40h	9:51
qslpt53	iop(4)	9Eh	quadart(14)	60h	9:52
qslpt54	iop(4)	9Eh	quadart(14)	60h	9:53
qslpt55	iop(4)	9Eh	quadart(14)	60h	9:54
qslpt56	iop(4)	9Eh	quadart(14)	60h	9:55
qslpt57	iop(4)	9Eh	quadart(15)	80h	9:56
qslpt58	iop(4)	9Eh	quadart(15)	80h	9:57
qslpt59	iop(4)	9Eh	quadart(15)	80h	9:58
qslpt60	iop(4)	9Eh	quadart(15)	80h	9:59
qslpt61	iop(4)	9Eh	quadart(16)	A0h	9:60
qslpt62	iop(4)	9Eh	quadart(16)	A0h	9:61
qslpt63	iop(4)	9Eh	quadart(16)	A0h	9:62
qslpt64	iop(4)	9Eh	quadart(16)	A0h	9:63

Device Definition for slpts
(TU-ART serial printers)
Character Devices

Device name	Board - Base port(hex)		Device number major : minor
slpt1	16fdc	00h	7:0
slpt2	TU-ART A	20h	7:2
slpt3	TU-ART B	50h	7:5
slpt4	TU-ART A	60h	7:6
slpt5	TU-ART B	70h	7:7
slpt6	TU-ART A	80h	7:8
slpt7	TU-ART B	90h	7:9
slpt8	TU-ART A	A0h	7:10
slpt9	TU-ART B	B0h	7:11

Device Definitions for tps (tape drives)

Device Name	IOP	Base Port	Device number (major:minor)
tp1	IOP1	CEh	11:0
tp2	IOP1	CEh	11:1
tp3	IOP1	CEh	11:2
tp4	IOP1	CEh	11:3
tp5	IOP1	CEh	11:4
tp6	IOP1	CEh	11:5
tp7	IOP1	CEh	11:6
tp8	IOP1	CEh	11:7
tp17	IOP2	BEh	11:16
tp18	IOP2	BEh	11:17
tp19	IOP2	BEh	11:18
tp20	IOP2	BEh	11:19
tp21	IOP2	BEh	11:20
tp22	IOP2	BEh	11:21
tp23	IOP2	BEh	11:22
tp24	IOP2	BEh	11:23
tp33	IOP3	AEh	11:32
tp34	IOP3	AEh	11:33
tp35	IOP3	AEh	11:34
tp36	IOP3	AEh	11:35
tp37	IOP3	AEh	11:36
tp38	IOP3	AEh	11:37
tp39	IOP3	AEh	11:38
tp40	IOP3	AEh	11:39
tp49	IOP4	9Eh	11:48
tp50	IOP4	9Eh	11:49
tp51	IOP4	9Eh	11:50
tp52	IOP4	9Eh	11:51
tp53	IOP4	9Eh	11:52
tp54	IOP4	9Eh	11:53
tp55	IOP4	9Eh	11:54
tp56	IOP4	9Eh	11:55

Device Definitions for Floppy Disk Drives
(Block Devices)

Device Name	Board	Device number major : minor
fda	16fdc	1:0
fdb	16fdc	1:1
fdc	16fdc	1:2
fdd	16fdc	1:3
sfda	16fdc	1:4
sfdb	16fdc	1:5
sfdc	16fdc	1:6
sfdd	16fdc	1:7

Device Definitions for Hard Disk Drives
(Block Devices)

Device Name	Board	Device number major : minor
hd0	wdi	2:0
hd1	wdi	2:1
hd2	wdi	2:2

Device Definitions for System Drivers

Device Name	Board	Block or Character	Device number major : minor
null		C	3:0
smem		C	3:1
timer			4:0
iomem1	iop(1) - CEh	C	8:0
iomem2	iop(2) - BEh	C	8:16
iomem3	iop(3) - AEh	C	8:32
iomem4	iop(4) - 9Eh	C	8:48
root		B	0:0

Appendix E

ASCII TABLE

00h	NUL	(▲@)	2Bh	+	56h	V
01h	SOH	(▲A)	2Ch	,	57h	W
02h	STX	(▲B)	2Dh	–	58h	X
03h	ETX	(▲C)	2Eh	.	59h	Y
04h	EOT	(▲D)	2Fh	/	5Ah	Z
05h	ENG	(▲E)	30h	0	5Bh	[
06h	ACK	(▲F)	31h	1	5Ch	\
07h	BEL	(▲G)	32h	2	5Dh	]
08h	BS	(▲H)	33h	3	5Eh	^
09h	HT	(▲I)	34h	4	5Fh	_
0Ah	LF	(▲J)	35h	5	60h	'
0Bh	VT	(▲K)	36h	6	61h	a
0Ch	FF	(▲L)	37h	7	62h	b
0Dh	CR	(▲M)	38h	8	63h	c
0Eh	SO	(▲N)	39h	9	64h	d
0Fh	SI	(▲O)	3Ah	:	65h	e
10h	DLE	(▲P)	3Bh	;	66h	f
11h	DC1	(▲Q)	3Ch	<	67h	g
12h	DC2	(▲R)	3Dh	=	68h	h
13h	DC3	(▲S)	3Eh	>	69h	i
14h	DC4	(▲T)	3Fh	?	6Ah	j
15h	NAK	(▲U)	40h	@	6Bh	k
16h	SYN	(▲V)	41h	A	6Ch	l
17h	ETB	(▲W)	42h	B	6Dh	m
18h	CAN	(▲X)	43h	C	6Eh	n
19h	EM	(▲Y)	44h	D	6Fh	o
1Ah	SUB	(▲Z)	45h	E	70h	p
1Bh	ESC	(▲[)	46h	F	71h	q
1Ch	FS	(▲\)	47h	G	72h	r
1Dh	GS	(▲])	48h	H	73h	s
1Eh	RS	(▲^)	49h	I	74h	t
1Fh	VS	(▲_)	4Ah	J	75h	u
20h	SPACE		4Bh	K	76h	v
21h	!		4Ch	L	77h	w
22h	"		4Dh	M	78h	x
23h	#		4Eh	N	79h	y
24h	$		4Fh	O	7Ah	z
25h	%		50h	P	7Bh	{
26h	&		51h	Q	7Ch	¦
27h	'		52h	R	7Dh	}
28h	(		53h	S	7Eh	~
29h	)		54h	T	7Fh	DEL
2Ah	*		55h	U		

Appendix F

DISK ERROR MESSAGES

In the event of a system malfunction, the Cromix Operating System displays a complete error message to aid in the diagnosis and correction of the problem. The following section describes these messages and their interpretation.

FLOPPY DISK ACCESS ERROR MESSAGES

When the operating system cannot successfully access a diskette an error message is displayed.

Format:

Disk Mode Error: Dev: maj dev: min dev; Blk #, Cylinder cc, Sector ss, Status=ee

where:

mode stands for one of the following words:

Seek	Error occurred in seeking a track on the disk.
Read	Error occurred during a read from the disk.
Write	Error occurred during a write to the disk.
Home	Error occurred in seeking track 0 on the disk.
Read-after-Write	Error occurred during the Cyclic Redundancy Check.

x is a letter from A to H which represents the disk drive with the error.

cc is the cylinder (in hexadecimal) where the error occurred.

ss is the sector number (in hexadecimal) where the error occurred.

ee is the 8 bit status byte displayed in hexadecimal which describes the error and the conditions at the time the error occurred.

The status byte is a hexadecimal number that is either one of the hex values in the table below or the combination or two or more of those hex values. The bits which correspond to those hex values describe the reasons or the error.

	Status Bits Set and Corresponding Hexadecimal Values							
Bits	7	6	5	4	3	2	1	0
Hex value	80	40	20	10	8	4	2	1

If the status byte were 0A, the bits set would be 3, 1, and 0 because the only combination of corresponding hexadecimal values that add up to 0A are the ones which correspond to bits 3, 1, and 0.

The following table describes the malfunctions corresponding to the bits set in the status byte.

Status Bits Set	Seek	Read	Write
7	not ready	not ready	not ready
6	write protect*	record type*	write protect
5	head engaged*	record type*	write fault
4	seek error	record not found	record not found
3	crc error	crc error	crc error
2	track 0*	lost data	lost data
1	index*	data request*	data request*
0	busy*	busy*	busy*

Status Bits Set	Home	R-A-W
7	not ready	not ready
6	write protect*	record type*
5	head engaged*	record type*
4	seek error	record not found
3	crc error	crc error
2	track 0*	lost data
1	index*	data request*
0	busy*	busy*

The asterisk (*) in the table above indicates that the condition is not the cause of the error message, but that it was present when the error occurred. For example, if the status byte was 30H during a Seek error, bits 4 and 5 are set (=1). This is a Seek error and the head is engaged. The head is supposed to be engaged during a seek. Therefore, this condition is not an error, and is marked with an asterisk. CRC stands for Cyclic Redundancy Check. It is a verification done after a Read or Read-after-Write operation. A CRC error indicates that data was not transferred without error.

```
Read Error, Drive B, Track 1C, Sector 10, Status=10
```

During a Read operation, status code 10 or 08 indicates the data is not readable. This may be caused by bringing the disk close to a magnetic source or by scratching or otherwise mishandling the disk.

HARD DISK ERROR MESSAGES

If the Cromix Operating System encounters an error when accessing a hard disk drive, it displays the error in the following format:

Disk Mode error: Dev: maj dev: min dev; Blk #, Cyl #,
Surf #, **d** Cylinder **cc** Surface **hh**
Sector **ss** Status **ffss**

where:

mode — is either Read error, Write error, Verify, Home error, or Seek error.

d	is the letter of the drive.
cc	is the number of the cylinder in hexadecimal.
hh	is the head number.
ss	is the sector number in hex.
ffss	is the error number. The first two digits indicate the fatal error number and the second two digits indicate the system error number.

Hard Disk Fatal Errors

The following error codes are displayed when a fatal disk error occurs:

00 Failed to Seek & Read Header during R/W

An error occurred during an attempt to seek & read the header preceding a read/write operation.

01 Failed to Seek - Timeout

The seek did not complete within a specified time. Check the drive electronics.

02 Fault Occurred during Seek

During the seek, a fault error occurred within the drive, as reported by the drive. This may be any of several errors.

03 Failed to Seek to Correct Track

The sector header as read off the disk is not what the drivers expected, thus the current disk location is incorrect.

04 Failed to Read CRC of Header

The CRC for the header as read from the disk is incorrect; it is different than what was expected. Most likely the current disk location is incorrect or the media surface is damaged.

05 Failed to Rezero - Timeout

A rezero command did not complete within a

specified time. Check the drive electronics.

06 Fault Occurred after Rezeroing

A fault error occurred within the drive after a rezero command was executed. This may be any of several errors.

07 Drive not Ready

The ready signal from the drive is not active. Make sure the drive is connected properly.

08 Failed to Write - Fault Error

During the write, a fault error occurred within the drive, as reported by the drive. This may be any of several errors.

09 Failed to Verify after Write

After data is written to the disk, it is read back and verified. This error occurs if the data cannot be properly verified.

0A Failed to Read - Fault Error

During the read, a fault error occurred within the drive, as reported by the drive. This may be any of several errors.

0B Failed to Read - CRC Error

The CRC just read from the disk is incorrect; it is different from the expected CRC. This error usually means that the data just read is incorrect.

0C Failed to Read - Cannot Locate Sector

The sector being looked for cannot be found on the current track. This error occurs if the media surface is damaged or if the controller electronics are not functioning properly.

0D Surface is Write Protected

The surface selected for the current write command is write protected and cannot be written to.

Hard Disk System Errors

The following error codes are displayed when a system disk error occurs:

00 No Acknowledge Received from Drive

The drive did not acknowledge a command sent to it. Make sure the drive is connected properly.

01 Drive Remains BUSY - Acknowledge Stuck Low

The acknowledge signal from the drive did not go high again after the command strobe went inactive.

02 Timeout Occurred during Rezeroing

A rezero command did not complete within a specified time. Check the drive electronics.

03 Fault Condition Reported by Drive

A fault condition occurred within the drive, as reported by the drive. This may be any of several errors.

04 Failed to Read - CRC Error

The CRC just read from the disk is incorrect; it is different from the expected CRC. This error usually means that the data just read is incorrect.

05 Header Off the Disk Does Not Compare with Expected Header

The sector header as read off the disk is not what the drivers expected, thus the current disk location is incorrect.

06 Failed to Verify after Write Operation

After data is written to the disk, it is read back and verified. This error occurs if the data cannot be properly verified.

023-4022

Cromemco™
incorporated
Tomorrow's Computers Today
280 BERNARDO AVE., MOUNTAIN VIEW, CA 94043